FSA Modernization Partner

United States Department of Education

Federal Student Aid

Integrated Technical Architecture Release 3.0 Technical Specification

Task Order #69

Deliverable # 69.1.3

Version 3.0

June 03, 2002

Table of Contents

1 Change Log

Suggested Changes/Comments	Page	Author	Date	Change Made Y/N	Comment
Added Change Log and made modifications based on IV&V feedback	12-130	Wayne Chang	05/24/2002	Y	Version 2.0
Updated Change Log and made modifications based on IV&V feedback	7 - 130	Roshani Bhatt	06/03/2002	Y	Version 3.0

2 Executive Summary

2.1 Introduction

The Integrated Technical Architecture (ITA) Release 3.0 (R3.0), like its previous releases, provides the Department of Education's Federal Student Aid (FSA) enterprise with a more robust core architecture for its application and production efforts. In addition to providing FSA's application teams with a core set of products and Subject Matter Expert (SME) support, ITA R3.0 will provide an additional set of Java 2 Enterprise Edition (J2EE) application architecture Reusable Common Services (RCS). The ITA Release 2.0 Strategic Assessment of the FSA applications and their usage of ITA R2.0 RCS, identified reusable common services as a means of providing significant value to FSA Java development efforts.

The first release of ITA RCS was well received and recognized as a valuable contribution towards Java application development efforts. During ITA R2.0 strategic assessment interviews, ITA RCS 2.0 received exceptionally positive feedback from the application development teams. To continue providing value added services, the ITA in its third release is offering additional RCS to FSA for its current and future developments initiatives.

This document provides an overview of designs of the ITA R3.0 Reusable Common Services. The reusable common services provide FSA with an additional set of Java services that may be used across the enterprise to handle Java application architecture functions.

The ITA R3.0 Technical Specification provides information on the following nine ITA reusable common services:

- Web Conversation
- Object Pooling
- User Session
- File Transfer Protocol (FTP)
- Configuration
- XML Helper
- JSP Tag Library
- Scheduler
- Web Services

2.2 Description of sections

The ITA R3.0 Technical Specification is divided into the following sections:

- Section 1 provides an overview of the document
- Section 2 provides a functional overview of the ITA Release 3.0 reusable common services
- Section 3 provides detailed designs for each reusable common service
- Appendix A provides a matrix that details the current and potential usage of reusable common services by FSA's applications

2.3 Approach

The common services are built based on open technology and J2EE architecture. The ITA team leveraged previous Accenture and industry experience to design and develop a set of application common services that specifically address FSA enterprise Java application requirements.

2.4 Purpose

This document provides an overview, feature list, and detailed technical designs for the ITA RCS frameworks. These frameworks compose a suite of frameworks provided for use in FSA applications by the ITA 3.0 initiative. The goal of the ITA initiative is to promote code reuse, standardization of development, ease of application maintenance, and application of best practices across all FSA application development projects.

2.5 Scope

The ITA Release 3.0 Technical Specification provides design information on the components that directly compose the ITA RCS frameworks. While the frameworks make use of many J2EE features and packages, such as JDBC, JSP, Servlets, and XML, the Technical Specification does not cover these topics. For additional information on these topics, please refer to the Sun Java website (http://www.javasoft.com) or applicable Java programming guides.

2.6 Assumptions

The RCS were built and tested in the ITA R3.0 environment. The following table summarizes the ITA R3.0 products and versions with which the RCS components interface. While the RCS services were built using these product versions, the services were built in accordance with J2EE standards and to support product upgrades.

Function	Product	ITA 3.0
Operating System	Sun Solaris	v. 2.6
HTTP Server	IBM HTTP Server	v. 1.3.12.2
Java Application Server	WebSphere Application Server Advanced Edition	v. 3.5.3
Search Engine	Autonomy Knowledge Server	v. 2.2
Database	Oracle 8i	v. 8.1.7
Java Development Tool	Visual Age for Java Enterprise Edition	v. 3.5.3

2.7 Intended Audience

The ITA R3.0 Technical Specification is intended for ITA and FSA application programmers who need to understand the RCS frameworks in order to troubleshoot or enhance the applications. The document is not intended as a programmer's guide for how to use the frameworks in developing applications. A user's guide detailing how to use the RCS frameworks will be provided in a separate document.

3 Functional Overview

This section provides a functional overview of the ITA Release 3.0 RCS.

3.1 Web Conversation Framework

The purpose of the ITA Web Conversation framework is to provide a standard for developers and designers to apply the Model View Controller (MVC) design pattern for architecting and developing Java web-based applications. This framework allows different tiers of the web application to be created independently of one another, provides the ability to update the static content displayed without updating and recompiling code, and facilitates internationalization of web pages. The Web Conversation Framework was developed based on the Jakarta Struts Framework.

There are several benefits to using Struts as the basis of the Web Conversation framework:

- Full separation of application flow, business objects, and presentation documents
- An existing web application structure that enables developers to focus on development of the application rather than the flow logic
- Simplified handling of user-provided form data
- Built-in configurable support for internationalization

3.2 Object Pooling Framework

Object pooling allows the reuse of instantiated Java objects – i.e., database connection pool. Objects that are used repeatedly throughout the life of a Java application can potentially benefit from this framework. Using the framework allows applications to create and manage pools of objects. Objects in the pool are retrieved when called and returned to the pool at the end of their usage. The framework allows application development teams to take advantage of object pooling to increase application performance.

Object Pooling Framework provides the following features:

- Managed object pools for easy retrieval and usage of objects
- Efficient use of system resources to increase application performance
- First In First Out (FIFO) object retrieval mechanism

3.3 User Session

The User Session simplifies, standardizes, and extends the use of user session / context information within the J2EE standard. The session wrapper class provides a common way to access session information, decouples session information from the request, HTTP session, and application J2EE session contexts, and wraps WebSphere session extension classes. The Session Framework provides the best practices and higher performance implementation for applications requiring storage of large amounts of session information.

The session framework will provide for the following services on both the client-side and server-side contexts:

- Providing a common interface to all HTTP variables (request, cookie, session)
- Storing temporary data that needs to persist across web pages

3.4 File Transfer Protocol

The purpose of the File Transfer Protocol (FTP) framework is to provide a standard interface to applications for transferring files. The framework allows applications to create FTP connections and execute a variety of FTP commands (which include the upload and download of files). A graphical user interface allows applications to easily add FTP functionality.

The FTP Framework provides the following features:

- Active and Passive transfer modes
- Implements javax.net.ssl package for secure file transfer
- User friendly graphical user interface
- APIs to be called within application code

3.5 Configuration

The configuration framework provides a standard for setting up and accessing application-wide configuration data. The framework allows configuration information to be loaded from properties files, XML files, or database tables. The configuration framework abstracts the representation of configuration data (the format in which they are written) so that if the files change formats, only the configuration framework will require coding changes – no coding changes will be needed within the application or architecture code.

The Configuration Framework is implemented using the Accenture's General and Reusable Netcentric Delivery Solution (GRNDS) configuration framework. The GRNDS code has been extended to meet FSA application development requirements. Specifically, the framework has been extended to:

- Use a static initializer to load the configuration files, instead of using the GRNDS bootstrap framework
- Support configuration input from database tables

3.6 XML Helper

The XML Helper framework will provide two functionalities. First, the XML framework will provide a parsing function for reading property files in XML format and setting up parameters for applications. Second, the framework will provide conversion function between XML and Java objects. With this framework, XML that describes a Java object can be constructed from the particular XML and a Java object can be mapped to an XML representation.

3.7 JSP Tag Library Framework

The JSP Tag Library Framework provides a collection of commonly used JSP custom tag libraries for application developers to implement. The JSP tag library framework is comprised of JSP Tags from the Apache Struts framework, external sources, and custom developed libraries. A JSP tag allows developers to package a commonly used set of code in an easy, reusable code library. Once built, the actual code library provides a simple set of custom tags that even a non-Java developer can use.

The JSP Tag Library offers the following taglibs:

- **Jakarta Struts Bean Taglib** - contains custom JSP used to define new beans from a variety of sources and to render a bean or bean property to the output response

- **Jakarta Struts HTML Taglib** - contains JSP custom tags useful in creating dynamic HTML user interfaces, including input forms

- **Jakarta Struts Logic Taglib** – contains tags useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management

- **Jakarta Struts Template Taglib** - contains tags that are useful in creating dynamic JSP templates for pages that share a common format

- **Jakarta DateTime Taglib** - contains tags that can be used to handle date and time related functions

- **Jakarta I18N Taglib** - contains tags that help manage the complexity of creating internationalized web applications

- **Jakarta Input Taglib** – contains tags that present HTML <form> elements that are tied to the ServletRequest that causes the current JSP page to run. Can be used to pre-populate form elements with prior values that the user has chosen or with default values for the first time web page visitors

- **Log Taglib** – contains tags that embed logging calls in JSP using the Logging Framework

- **Jakarta Page Taglib** – contains tags that can be used to access all of the PageContext information of a JSP page within the context of the current page

- **Jakarta XSL Taglib** – contains tags used to process XML documents with an XSL stylesheets and inserts them in place.

- **XTags Taglib** – contains custom tags for working with XML and implements an XSLT-like language allowing XML to be styled and processed from directly within a JSP

3.8 Scheduler

The Scheduler provides the capability for applications to execute pre-determined tasks automatically and periodically. The Scheduler framework will be developed to meet demand of FSA application teams. Scheduled emailing, FTP and database maintenance functions are examples for which applications routinely perform either manually or by Unix cron jobs. Setting Unix cron jobs and Unix scripts to execute Java classes can be confusing and unreliable. The Scheduler provides a user-friendly interface and the mechanism for scheduling Java-based tasks.

3.9 Web Services

Consists of SOAP, XML, and UDDI technologies to enable web services. Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. The invocation is done using a set of low overhead, open standard network and application protocols (i.e., UDDI, SOAP, XML, etc). The Web Services framework describes the technical architecture of a Web Service. It also provides an implementation for the SOAP protocol put forth by the World Wide Web Consortium (W3C).

4 Detailed Designs

This section provides detailed designs for the ITA Release 3.0 RCS frameworks.

4.1 Web Conversation Framework

4.1.1 Purpose

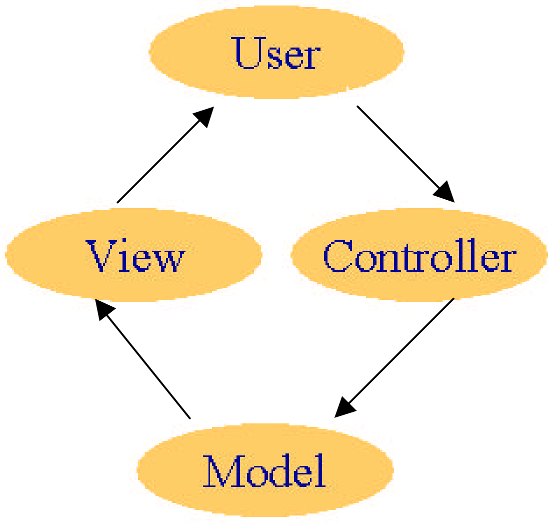
The purpose of the ITA Web Conversation framework is to provide a standard for developers and designers to apply the MVC (Model View Controller) design pattern for developing web applications. This framework allows different tiers of the web application to be created independently of one another, provide the ability to update the static strings displayed without updating and recompiling code, and facilitates internationalization of web pages.

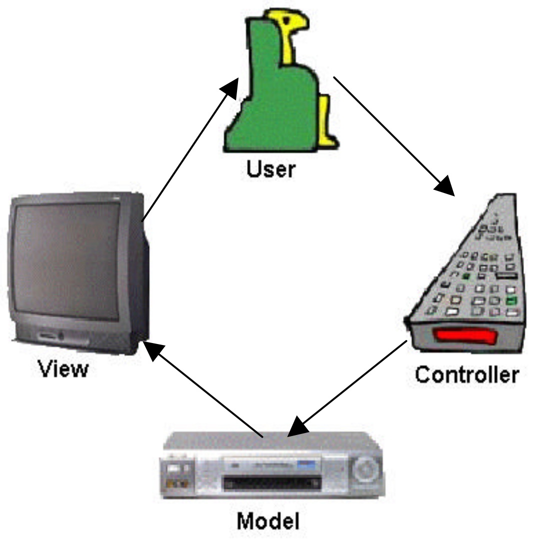
4.1.2 System Overview

The Web Conversation framework is based on the Jakarta Struts. The Struts framework enables developers to combine the use of JavaServer Pages (JSP) and Java Servlets to create dynamic pages. Struts is part of the Jakarta Project, sponsored by the Apache Software Foundation (http://jakarta.apache.org/). The Apache Software Foundation provides support for the Apache community of open-source software projects. The Apache projects are characterized by a desire to create high quality software that leads the way in its field, an open and pragmatic software licensing and a collaborative, consensus based development process.

Struts is an open source framework that takes advantage of the MVC design pattern to enable developers to focus on one portion of the process without having to know details of how the other components work. The Struts framework provides many objects that facilitate the fundamental aspects of MVC, while allowing the developer to extend the design and functionality further as required by the application requirements. Struts allows default information to be configured in a file that can be updated at any time versus having to use static constants in the code.

The MVC design pattern is a technique used to separate business logic/state (the Model) from User Interface (the View) and program progression/flow (the Control). The user is presented with the View, which in a web-based application can be an HTML page or an image. The View interacts with the Controller, which is responsible for the flow of the program and processes updates from the Model to the View and vice versa. The Model represents the business logic or state (data in the system) and is usually defined as a Java object. In a real-world situation, the View can be thought of as a TV Screen, the Model is the VCR/DVD player, and the Controller is the remote control.





Struts implements the three major components found in the MVC design pattern. Its primary function is to facilitate the use of JavaServer Pages for generating the HTML, while the Servlets are used to mediate the control flow using the MVC design pattern.

There are several benefits to using Struts as the basis of the Web Conversation framework:

- Full separation of application flow, business objects, and presentation documents
- An existing web application structure that enables developers to focus on development of the application
- Simplified handling of user-provided form data
- Built-in configurable support for internationalization
- Lower development cost due to minimum technical requirements to develop view, which can be constructed in HTML

4.1.3 Design Considerations

4.1.3.1 Assumptions and Dependencies

The Web Conversation framework will function in a J2EE application server environment. The current distribution of Struts used is Struts 1.0.1. As the current production server for FSA is IBM's WebSphere Application Server (WAS) v. 3.5, the framework will be compiled using its required JDK version 1.2.2. It is also compatible with the current JavaServer Pages (1.1) and Java Servlet (2.2) specifications for this server. It is built and tested on the Sun Solaris 2.6 and HP-UX 11.0 operating systems.

While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.1.3.2 Goals and Guidelines

The Web Conversation framework will provide a robust framework that can easily be utilized by any FSA development team building web-based applications using Java Server Pages and Java Servlets. The foundation of the Web Conversation framework is based on Struts, which supports separation of application flow (Controller), business logic (Model) and presentation (View). This separation allows developers to build the programming logic while web designers concentrate on creating the front-end presentation pages.

4.1.3.3 Development Methods

This framework will be used as the basis for developing all web-based applications for the FSA project. The standard class and sequence diagrams are provided in this document in order to illustrate the framework's structure. These diagrams should assist developers who are unfamiliar with this framework.

4.1.4 System Architecture

The following sections will illustrate the Model-View-Controller (MVC) design pattern and define the MVC components as used in the web conversation framework.

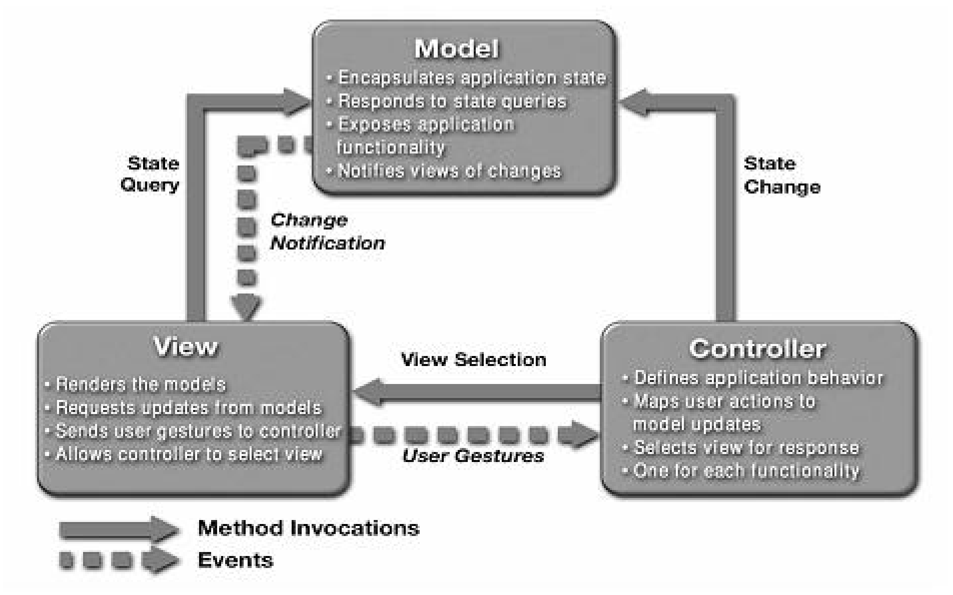

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Figure 1: MVC Design Pattern (Model 2)

There are three main components to Struts:

1. View – The View component in Struts is represented by JavaServer Pages and HTML. The View is completely independent of the underlying data model and business logic, allowing web authors to modify the presentation of content without affecting the underlying code. Struts also supports internationalization. It allows a web application to be rendered in different languages through the use of the application *resources.properties* file.

2. Model – The Model in Struts is usually defined as a Java object, or bean. Model classes should be coded independently of the Struts framework to promote maximum code reuse by other applications. The Model typically represents the underlying data and is independent of the presentation (view).

    - The Struts framework also provides some default Model components, most important of which is *ActionForm*. Struts ActionForms ensure the form bean is created when needed and the POST request from the submit action directly updates the form object with the inputted values, and ensures any included validation is run against the data. The ActionForm also ensures the form bean is passed to a Controller object. Struts will only handle Models in an automatic fashion if the **org.apache.struts.action.ActionForm** Class is extended

3. The Controller – The controller is the switchboard of MVC in Struts. It directs the user to the appropriate Views and provides the View with the correct Model. The task of directing users to appropriate Views is called "mapping". In Struts, the Controller class, ActionServlet, uses a configuration file called struts-config.xml to read in mapping data called ActionMapping. By reading this data, ActionServlet class can match the incoming Uniform Resource Identifier (URI) against a set of ActionMappings to find an Action class that handles the request. This process is described in the following diagram:

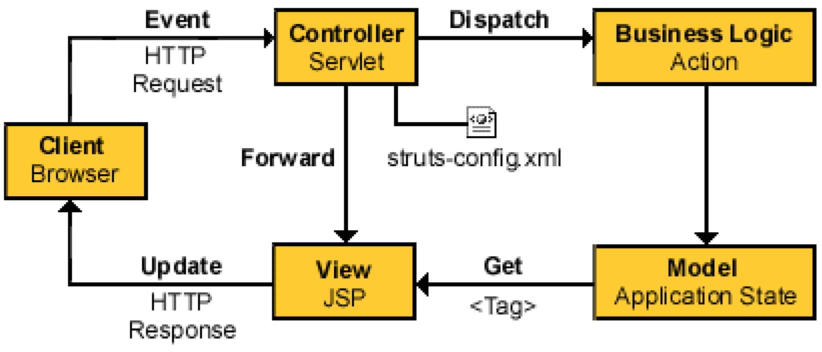


Figure 2: Struts Framework

In the Figure 2 above, ActionServlet reads the struts-config.xml file that contains the mapping of actions to determine which object to call for a given Action request. The file is read at startup of the application and the relationships are stored in memory for optimal performance. ActionServlet responds to HTTP Requests and direct each request the appropriate Action. Actions provide further direction to the business logic and will update and validate the ActionForm - if necessary. The JSP accesses the updated ActionForm and the output is sent as the HTTP Response to the browser.

4.1.5 Detailed System Design

4.1.5.1 Component Overview

The figure below illustrates the Struts framework, the interaction between the various components and demonstrates how the Web Conversation framework maps into the MVC design pattern.

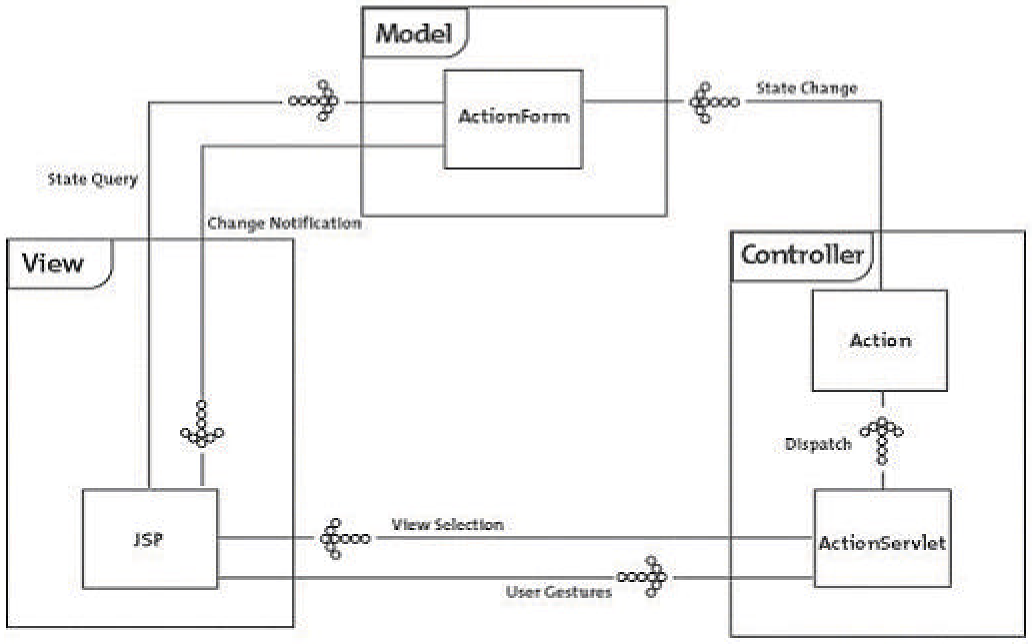


Figure 3: The Struts Framework in MVC

In the Struts framework, the client browser sends an HTTP Request to the Controller. The ActionServlet Controller component reads from the struts-config.xml for the Action to call, and from the Application resources.properties file for any strings that are pre-defined, and is also used for internationalization (i18n). The ActionServlet then passes control to the Action Controller listed in the struts-config.xml file. The Action calls the ActionForm (the Model) component, which process the business logic of the application and performs any necessary updates and validation. The Model will return an error or return control directly to the View object.

4.1.5.2 Component Definitions

4.1.5.2.1 JavaServer Pages

JavaServer Pages (JSP) serve as the View component in the Struts framework. The JSP contain static HTML and offers authors the ability to insert dynamic content based on the interpretation (at page request time) of special action tags. Struts includes a custom tag library that facilitates creating user interfaces that interact with ActionForm beans (part of the Model component).

The example below is of a JSP login form which uses JSP Taglibs:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<html:html locale="true">
<head>
<title><bean:message key="logon.title"/></title>
<html:base/>
</head>
<body bgcolor="white">
<html:errors/>

<html:form action="/logon.do" focus="username">
<table border="0" width="100%">
  <tr>
    <th align="right">
      <bean:message key="prompt.username"/>
    </th>
    <td align="left">
      <html:text property="username" size="16" maxlength="16"/>
    </td>
  </tr>
  <tr>
    <th align="right">
      <bean:message key="prompt.password"/>
    </th>
    <td align="left">
      <html:password property="password"
             size="16" maxlength="16"/>
    </td>
  </tr>
  <tr>
    <td align="right">
      <html:submit property="submit"
             value="Submit"/>
    </td>
    <td align="left">
      <html:reset/>
    </td>
  </tr>
</table>
</html:form>
</body>
</html:html>
```

4.1.5.2.2 Java Object

A Java Object (typically a Java Bean) usually represents the Model component in the Struts framework. The bean will represent details of the internal state of the system. The Model is made up of the abstract class ActionForm, which is a standard JavaBean with Get and Set methods that are used to access its state. All form classes that are automatically updated and validated within Struts extend org.apache.struts.action.ActionForm.

The following code is an example of an ActionForm bean:

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.upload.FormFile;

public class LogonForm extends ActionForm {
        protected String userName;
        protected String password;

        public void setUserName(String userName) {
                this.userName = userName;
        }
        public void setPassword(String password) {
                this.password = password;
        }
        //There would also be getters for these properties
        public void getUserName(String userName) {
                this.userName = userName;
        }
        public void getUserPassword(String password) {
                this.password = password;
        }
}
```

4.1.5.2.3 Servlets

The Controller is made up of two Struts classes, ActionServlet and Action. The ActionServlet class maps requests to specific Actions and is configured by defining a set of ActionMappings. The ActionServlet interacts with the Model, encapsulates the business logic, interprets the outcome, and dispatches control to the View to create the response.

This is a Controller example of an Action:

```
public class LogonAction extends Action {

        public ActionForward perform(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response) {

                LogonForm myForm = (LogonForm) form;

                if (myForm.getUserName().equals(“john”) &&
                        myForm.getPassword().equals(“doe”)) {
                        //return a forward to our success page
                        return mapping.findForward(”success”);
                } else {
                        ActionErrors errors = new ActionErrors();
                        errors.add("password",
                                new ActionError("error.password.required"));
                        this.saveErrors(errors); //Action implements this method
                        //go back to the page that made the request
                        return (new ActionForward(mapping.getInput()));
                }

        }
}
```

4.1.5.2.4 web.xml

The web.xml file is the standard web application deployment descriptor, which must contain a Servlet definition for the Struts Action Servlet. The web.xml file is read when the JSP container starts. An example of a web.xml file is shown here:

```
<web-app>
  <servlet>
<!--
Declare the O’ReillyAction servlet to be of type ActionServlet from the framework. Include the
configuration file as defined in the config param. Set the debug level to 2 with a detail of 2 when loading
the servlet. Create 2 instances.
-->
    <servlet-name>OreillyActionServlet</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
<!--
  All incoming requests that end in .action, send to the OreillyActionServlet.
-->
<servlet-mapping>
    <servlet-name> OreillyActionServlet </servlet-name>
    <url-pattern>*.action</url-pattern>
  </servlet-mapping>
<!--
  Send initial requests to the login page for this application
-->
<welcome-file-list><welcome-file>login.jsp</welcome-file></welcome-file-list>
<!--
  Make all of the necessary related Struts JSP custom tag libraries
available and define where to find them.
-->
  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
</web-app>
```

4.1.5.2.5 struts-config.xml

The main control file in the Struts framework is the struts-config.xml file. This file defines action mappings. The XML structure is defined by the struts-config Document Type Definition (DTD) file and can be found in the /docs/dtds subdirectory of the framework's installation root directory. The DTD contains the list of legal elements that can be used in the XML document. The top-level element is struts-config and it consists of the following elements: Data-source, form-bean, global-forwards, and action-mappings.

This is a partial example of a struts-config.xml file, other Actions identified in the DTD have been left out:

```
<struts-config>
  <form-beans>
    <form-bean
      name="logonForm"
      type="org.apache.struts.example.LogonForm" />
  </form-beans>
  <global-forwards
      type="org.apache.struts.action.ActionForward" />
    <forward name="logon" path="/logon.jsp"
         redirect="false" />
  </global-forwards>
  <action-mappings>
    <action
        path="/logon"
        type="org.apache.struts.example.LogonAction"
        name="logonForm"
       scope="request"
       input="/logon.jsp"
     unknown="false"
    validate="true" />
  </action-mappings>
</struts-config>
```

4.1.5.3 Internationalization

Internationalization is achieved through the use of PropertyResourceBundles, which are classes that support String data. Instead of creating a class for each language supported by the application, text files can be used. The resource property file contains locale-specific set of static strings used for internationalization.

The properties file contains strings in the form of a unique key-value pair. A properties file can be created for each supported locale with a different International Organization for Standardization (ISO) language code value. Using this method, the same key referenced in the application code can be used in each locale file but the values can be different depending on the locale.

In the web.xml file, the initialization parameter “application” is used to reference the name and location of the resource files. In the example web.xml file below, when the ActionServlet is loaded it will read its application parameter to determine that it should load its string resources from the MyResources.properties file.

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
                org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
                <param-name>application</param-name>
                <param-value>MyResources</param-value>
        </init-param>
</servlet>
```

The ActionServlet will also determine the language the client is using. If the language used by the client is different from the default, the ActionServlet looks up the two-character ISO language code and tries to look for a properties file by appending the ISO code to the name of the class.

For example, if the locale of the client is French and the default is English, the ActionServlet would first look for the file in French (MyResources_fr.properties). If it is not able to find the file in French, it then looks for the file in English (MyResources.properties).

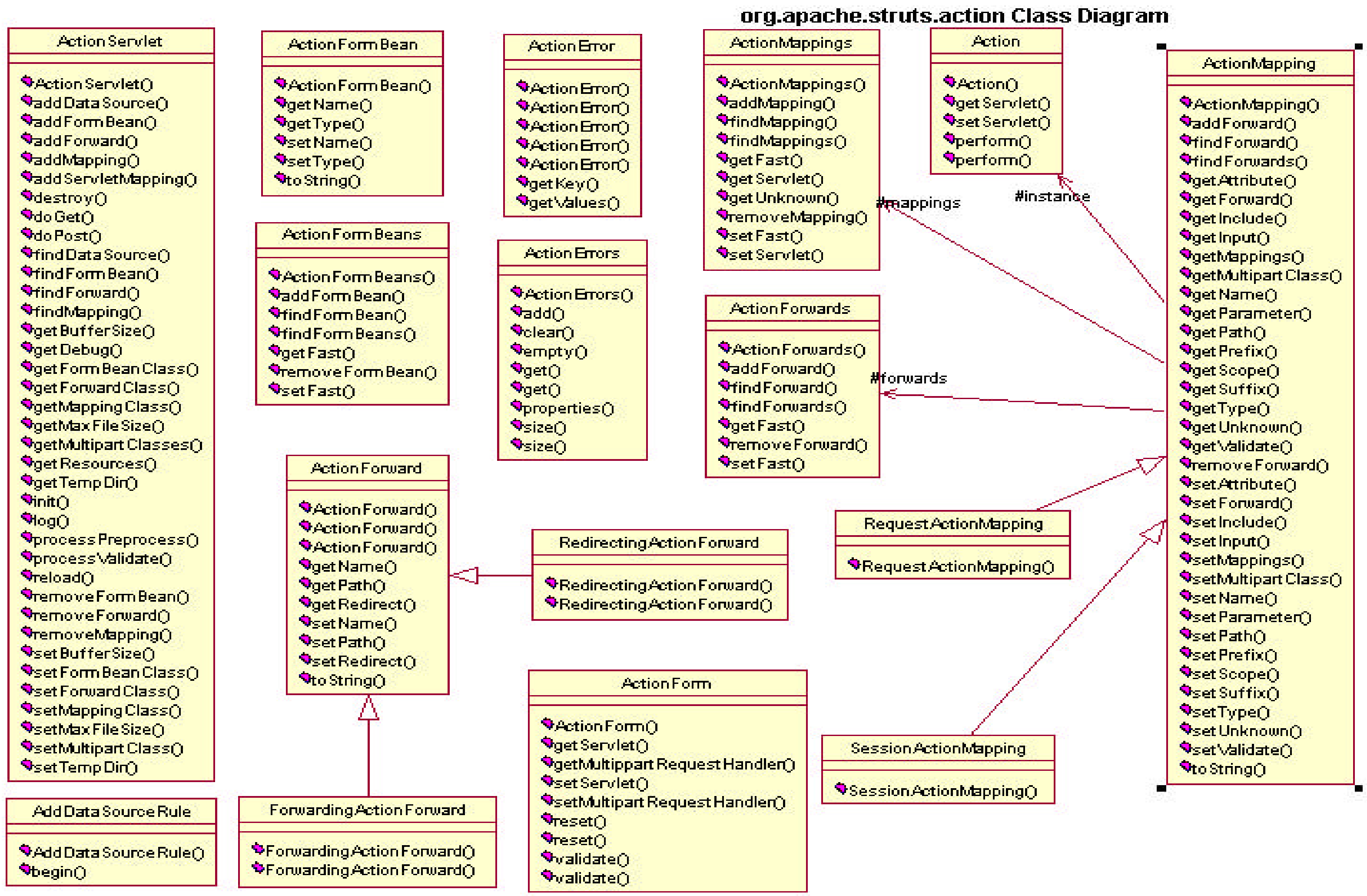
French: MyResources_fr.properties file:

```
helloworld.title=Allô monde!
menu.file=Dossier
menu.file.open=Ouvrir
menu.file.close=Fermer
menu.file.exit=Sortie
menu.edit=Rédiger
menu.edit.copy=Copier
menu.edit.paste=Pâte
```

Default: MyResources.properties file:

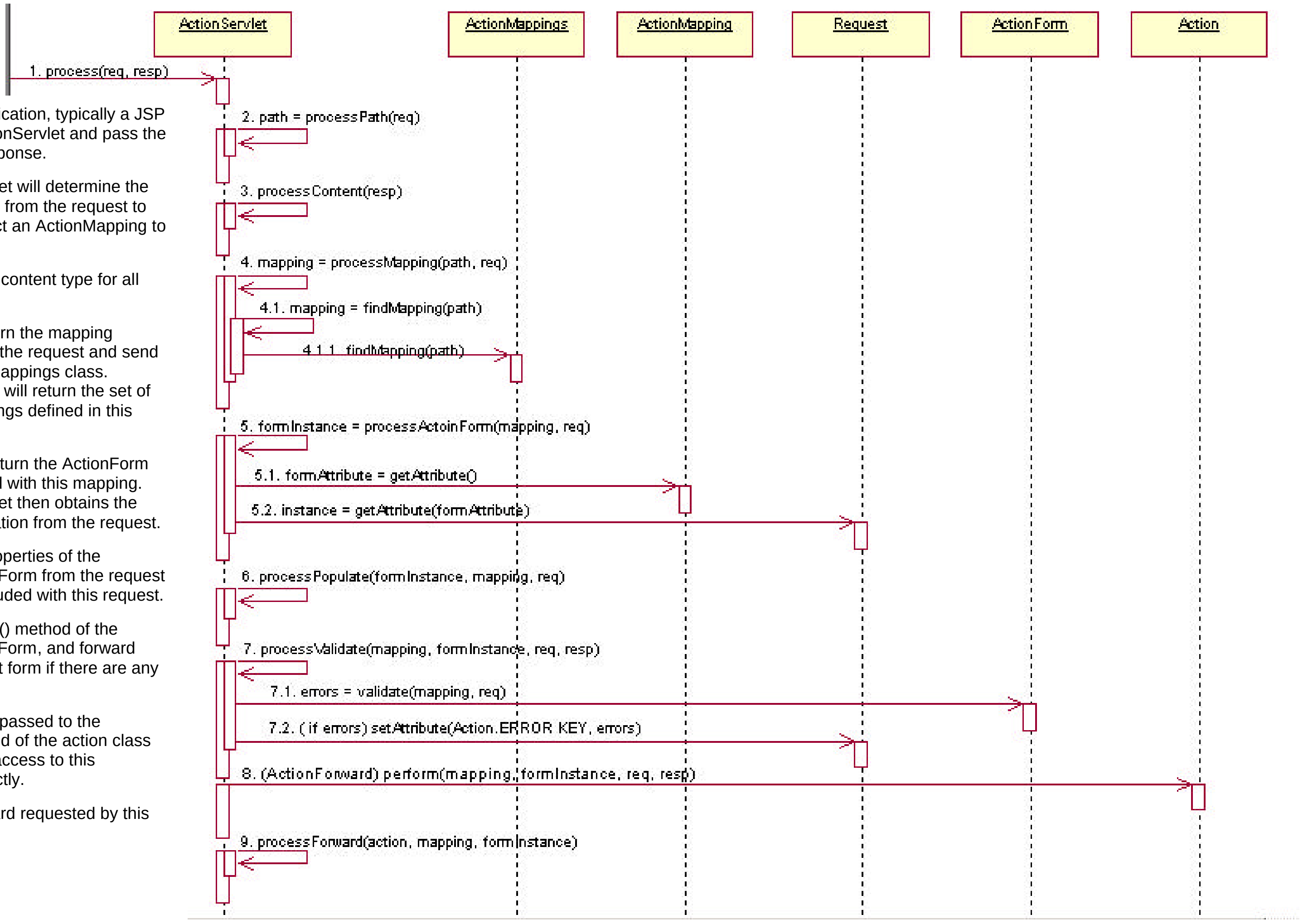
```
helloworld.title=Hello World!
menu.file=File
menu.file.open=Open
menu.file.close=Close
menu.file.exit=Exit
menu.edit=Edit
menu.edit.copy=Copy
menu.edit.paste=Paste
```

4.1.6 Class Diagram



4.1.7 Sequence Diagram

1) The calling application, typically a JSP will call the ActionServlet and pass the request and response.
2) The ActionServlet will determine the path component from the request to be used to select an ActionMapping to dispatch.
3) Sets the default content type for all responses.
4) Identify and return the mapping associated with the request and send it to the ActionMappings class. ActionMappings will return the set of paths for mappings defined in this collection.
5) Retrieve and return the ActionForm bean associated with this mapping. The ActionServlet then obtains the attribute information from the request.
6) Populate the properties of the specified ActionForm from the request parameters included with this request.
7) Call the validate() method of the specified ActionForm, and forward back to the input form if there are any errors.
8) The mapping is passed to the perform() method of the action class itself, enabling access to this information directly.
9) Process a forward requested by this mapping, if any.


ActionServlet
ActionMappings
ActionMapping
Request
ActionForm
Action
1. process(req, resp)
2. path = processPath(req)
3. processContent(resp)
4. mapping = processMapping(path, req)
4.1. mapping = findMapping(path)
4.1.1. findMapping(path)
5. formInstance = processActoinForm(mapping, req)
5.1. formAttribute = getAttribute()
5.2. instance = getAttribute(formAttribute)
6. processPopulate(formInstance, mapping, req)
7. processValidate(mapping, formInstance, req, resp)
7.1. errors = validate(mapping, req)
7.2. (if errors) setAttribute(Action.ERROR KEY, errors)
8. (ActionForward) perform(mapping, formInstance, req, resp)
9. processForward(action, mapping, formInstance)

4.1.8 References

- Struts Homepage

  http://jakarta.apache.org/struts
- Struts Documentation - Apache Struts Framework (Version 1.0)

  http://jakarta.apache.org/struts/api-1.0/index.html
- Struts, an open-source MVC implementation

  http://www-106.ibm.com/developerworks/java/library/j-struts/
- Strut Your Stuff with JSP Tags: Use and extend the open source Struts JSP tag library

  http://www.javaworld.com/javaworld/jw-12-2000/jw-1201-struts.html
- Introduction to Jakarta Struts Framework – Parts 1 – 3

  http://www.onjava.com/lpt/a//onjava/2001/09/11/jsp_servlets.html
  http://www.onjava.com/pub/a/onjava/2001/10/31/struts2.html
  http://www.onjava.com/pub/a/onjava/2001/11/14/jsp_servlets.html
- The Struts Framework's Action Mappings Configuration File

  http://www.informit.com/content/index.asp?product_id={0917F29F-56D8-4B25-9C67-211EC945BBAB
  (Requires creating a free informit.com account.)
- Building a Web Application: Strut by Strut

  http://husted.com/about/scaffolding/strutByStrut.htm
- Java Developer's Journal

  http://www.sys-con.com/java/article.cfm?id=1175
- "Introduction to MVC and the Jakarta Struts Framework," Craig W. Tataryn

  http://www.computer-programmer.org/articles/struts/
- "J2EE FrontEnd Technologies: A Programmer's Guide to Servlets, JavaServer Pages, and Enterprise JavaBeans, " Lennart Jörelid

4.2 Object Pooling Framework

4.2.1 Purpose

The purpose of the object pooling framework is to provide a standard to simplify, and extend the use of object pooling within the J2EE standard. The pooling framework will provide a common way for developers to re-use objects from a pool of available objects. This lessens the overhead on the server every time an object is created or destroyed. Objects are instantiated in a pool upon startup and remain available to applications in the resource pool. When a client requires the use of the object, the object can be obtained from the pool and released back to the pool when it is no longer needed. This will increase the load time at startup but will decrease the time needed to obtain an object, thereby minimizing garbage collection activity.

4.2.2 System Overview

The pooling framework is created using the Java programming language. The pooling framework provides a mechanism for developers to access a common resource pool of objects rather than repeatedly instantiating and destroying commonly used objects. The framework creates one common interface for application developers to create the pool, manage objects in the pool, obtain objects from the pool, and release objects back to the pool. The framework allows developers to create ‘pool managers’ that can each track multiple pools of different object types.

The FSAPoolMgr class is the main class that application developers will use to create pools and to obtain and release objects from the specified pool. The Hashtable used by the pool manager class will be able to track the different pools available.

Application developers extend the abstract FSAPoolableObject class to create new pool types that can be pooled. If an object needs to be added to a pool, the class has to extend this class to become poolable.

Factory classes must implement the FSAPoolableFactory interface in order to create new objects to be included in the pool using the Component Factory framework.

The FSAObjectPool class builds a pool to hold the specified object that can be pooled. The user specifies the minimum and maximum size of the pool along with the timeout period in a configuration file. The amount of time to wait for an object from the pool before timing out is also defined in the constructor. The objects in the pool will be accessible through a queue. The first available object in the pool queue is provided to the first application waiting for an object from the pool in a ‘First in, first out’ (FIFO) pattern.

The FSAPoolException class will provide exception catching and handling for the object pooling framework. This class extends the existing RCS Exception Handling framework.

4.2.3 Design Considerations

4.2.3.1 Assumptions and Dependencies

The pooling framework will function in a J2EE compliant application server environment. As the current production servers for FSA are running IBM's WAS v. 3.5.3 and v. 3.5.5, the framework will be compiled using its required JDK version 1.2.2 – the version included within the ITA Application Server WAS 3.5. It will also work with the current Java Server Pages (1.1), Java Servlet (2.2), and Java Database Connectivity (2.0) specifications for this server. It will be fully tested with both the Sun Solaris 2.6 and the HP-UX 11.0 operating systems. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.2.3.2 Goals and Guidelines

The goal of the pooling framework is to provide a simple yet robust framework that can be easily utilized by any FSA application team developing in a Java environment. The pooling framework provides the developer with the ability to access a shared pool of instantiated objects, which improves application performance.

Creating an object pool is recommended if the application frequently creates and destroys large amount of objects in order to minimize overhead. This is the case with accessing a database; many connection objects are created and destroyed. Object pooling is not recommended if the same object type is not created and destroyed repeatedly.

4.2.3.3 Development Methods

This framework will be developed using general object-oriented software development techniques. The standard class and sequence diagrams are provided in this document. These diagrams should assist developers who are unfamiliar with this framework.

4.2.4 System Architecture

4.2.4.1 Overview

The pooling framework provides the ability for application developers to access objects from a pool of available objects rather than constantly creating and destroying objects. The initial load time is longer because all of the objects are instantiated on startup. There is no additional load time as each process does not need to instantiate a new object, and there is reduced garbage collection since objects are being returned to a pool instead of being destroyed.

4.2.4.2 Subsystem Architecture: FSAPoolMgr

Application developers call the FSAPoolMgr class to create new pools and to track existing pools.

4.2.4.3 Subsystem Architecture: FSAPoolableResource

The Abstract class is extended by developers to make the new class poolable. This class ensures that the new class type will have the ability to release the object and to test if the object is alive.

4.2.4.4 Subsystem Architecture: FSAPoolableFactory

Creates new objects to be added to a pool using the Component Factory framework.

4.2.4.5 Subsystem Architecture: FSAObjectPool

Class used to create the object pool with the minimum number of objects defined. Places the objects in a FIFO queue.

4.2.4.6 Subsystem Architecture: FSAPoolException

FSAPoolException is thrown when an error is encountered in the pooling framework and uses the Exception Handling framework.

4.2.5 Detailed System Design

4.2.5.1 Component Definitions

4.2.5.1.1 FSAPoolMgr

Class Name:	FSAPoolMgr
Component:	Pooling
Description:	Public class
Package:	gov.ed.fsa.ita.pooling
Superclass:	Object

Attribute	Type	Description
Private:		
m_poolListHash	java.util.Hashtable	Hashtable of the list of pools managed by this instance.

Con/Destructors	**Arguments** (Type, Name)	**Description**
FSAPoolMgr	none	The constructor method that creates the Hashtable that the Pools are stored in. The pool name is the key and the pool object is the value.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
createPool	String poolName, int minSize, int maxSize, int timeOutInterval,, FSAPoolableObject pObj)	void throws FSAPoolException	Create a pool and store the pool name and object into the Hashtable.
getPooledObject	String poolName	FSAObjectPool throws FSAPoolException	Obtains a pool object from the specified Hashtable based on the provided pool name.

4.2.5.1.2 FSAPoolableFactory

Interface Name:	FSAPoolableFactory
Component:	Pooling
Description:	Allows user-defined factory to create new object types that can be used in a pool will implement this factory class.
Package:	gov.ed.fsa.ita.pooling
Superinterface:	None

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
createPoolableObject	none	FSAObjectPool throws FSAPoolException	This empty method ensures an application developer uses this class to create a new Pooled Object.

4.2.5.1.3 FSAPoolableResource

Class Name:	FSAPoolableResource
Component:	Pooling
Description:	This abstract class will be extended to create new class types that can be pooled.
Package:	gov.ed.fsa.ita.pooling
Superclass:	Object

Attribute	Type	Description
Private:		
M_pool	gov.ed.fsa.ita.pooling.FSAObjectResource	Used to store the reference to the pool.

Con/Destructors	Arguments (Type, Name)	Description
FSAPoolableResource	none	This is the default constructor.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Protected:			
IsAlive	none	Boolean state	Tests to see if the object is still alive.
Public:			
Release	none	void	Returns the object back to the pool from where it came.
makeObjectPoolable	none	void – throws FSAPoolException	Makes the class into a class that can be pooled.

4.2.5.1.4 FSAObjectPool

Class Name:	FSAObjectPool
Component:	Pooling
Description:	This class creates the pool that objects are stored in.
Package:	gov.ed.fsa.ita.pooling
Superclass:	Object

Attribute	Type	Description
Private:		
m_poolName	String	Name of the pool.
m_minSize	int	Initial and minimum size of the pool.
m_maxSize	int	Maximum size of the pool. Objects will not be initialized and added to the pool beyond this number.

Attribute	Type	Description
m_timeOut	int	Setting for when timeout should occur if an object in a pool is not immediately available. • Set to -1 if there is no object, returns an error immediately • Set to 0 if wait indefinitely • Any positive value greater than 0 is the amount of time to wait in milliseconds before a retry is attempted. Will be retried three times using the specified wait time
m_poolSize	int	Value to hold the current size of the pool.
m_objectPool	FSAPoolQueue	The queue is used to hold the object pool and allow a FIFO pattern to be applied to the use of the objects in the pool.
m_requestQ	edu.oswego.cs.dl.util.concurrent.FIFOSemaphore	The Queue used to store requesting objects in.
m_retryAttempt	int	Number of retries to attempt before throwing an exception.
Con/Destructors	**Arguments** (Type, Name)	**Description**
FSAObjectPool	String name, int minSize, int maxSize, int timeOut	Create a new pool with the default values.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Protected:			
destroy	none	void	Destroys the object and removes it from the pool and queue if the object is no longer alive.
Public:			
getMaxValue	none	int	Obtains the maximum value the pool can hold currently.
getMinValue	none	int	Obtains the minimum value the pool can hold currently.
getRetryValue	none	int	Obtains the current retry value.
setMaxValue	int value	void	Sets the maximum value the pool can hold.
setMinValue	int value	void	Sets the minimum value the pool can hold.
setRetryValue	int value	void	Sets the number of retries that should be attempted.
createNewObject	none	FSAPoolableObject throws FSAPoolException	Create a new instance of the object and add it to the pool.
release	FSAPoolableObject	void	Returns the object to the pool.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
getPooledObject	none	FSAPoolableObject throws FSAPoolException	Gets an object from the pool.

4.2.5.1.5 FSAPoolException

Class Name:	FSAPoolException
Component:	Pooling
Description:	FSAPoolException will be thrown when errors are encountered while using the pooling framework.
Package:	gov.ed.fsa.ita.pooling
Superclass:	gov.ed.fsa.ita.exception

Con/Destructors	Arguments (Type, Name)	Description
FSAPoolException	none	Default Constructor with no error message provided.
FSAPoolException	String errMsg	Default Constructor with an error message.
FSAPoolException	String errMsg, Throwable e	Default Constructor with an error message and Throwable.

4.2.5.1.6 FSAPoolQueue

Class Name:	FSAPoolQueue
Component:	Pooling
Description:	This class stores the pool objects in a queue and uses a FIFO pattern to manage the use of the object in the queue.
Package:	gov.ed.fsa.ita.pooling
Superclass:	Object

Attribute	Type	Description
Protected:		
M_head	FSAPoolQueue.Node	The first entry in the queue.
M_tail	FSAPoolQueue.Node	The last entry in the queue.
M_getLock	Object	Synchronization target for poll/get operations.
M_putLock	Object	Synchronization target for put operations.

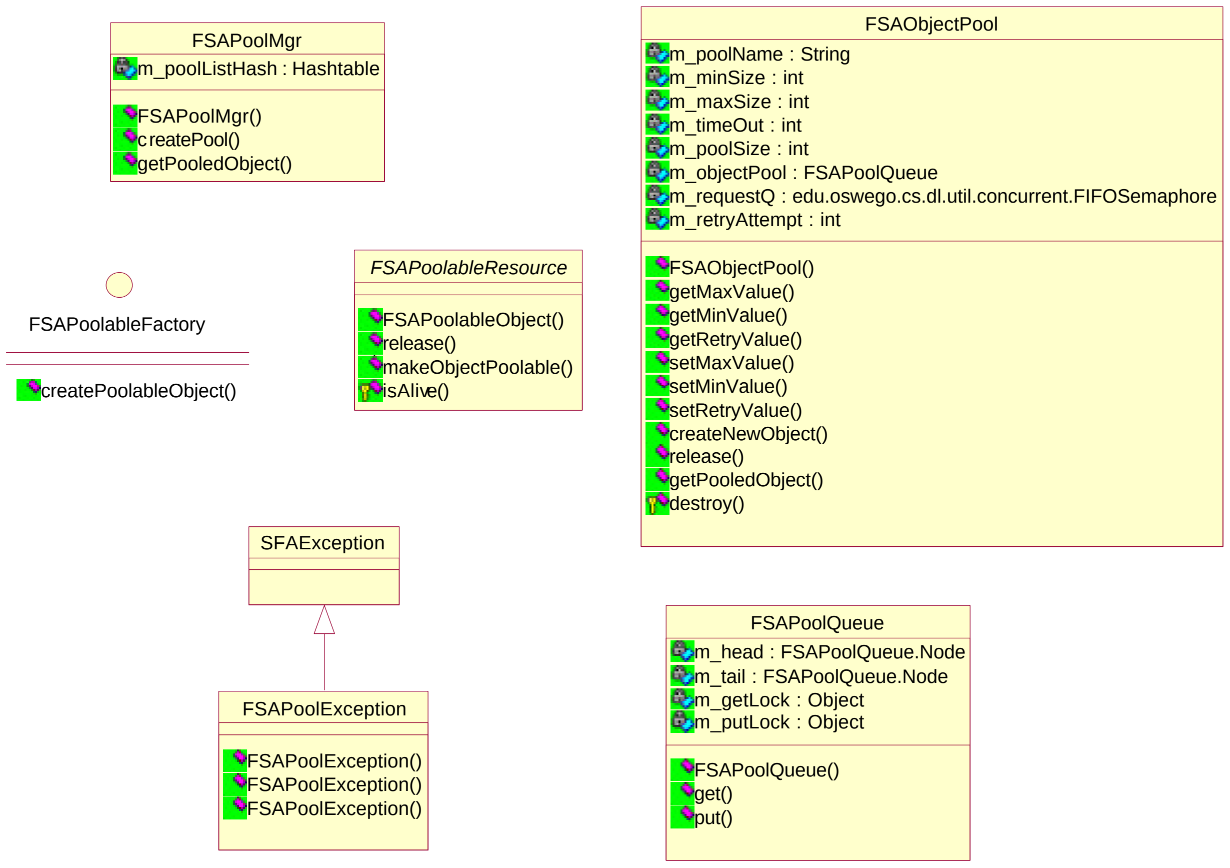
Con/Destructors	Arguments (Type, Name)	Description
FSAPoolQueue	none	Default constructor.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
Get	none	FSAPoolableObject throws FSAPoolException	Obtains and removes the first object from the head / front of the queue.
Put	FSAPoolableObject	void	Adds an object to the tail / end of the queue.

Class	**Description**
Node (private static)	private static class declared within the FSAPoolQueue class to act as a placeholder to reference the next object in the queue.

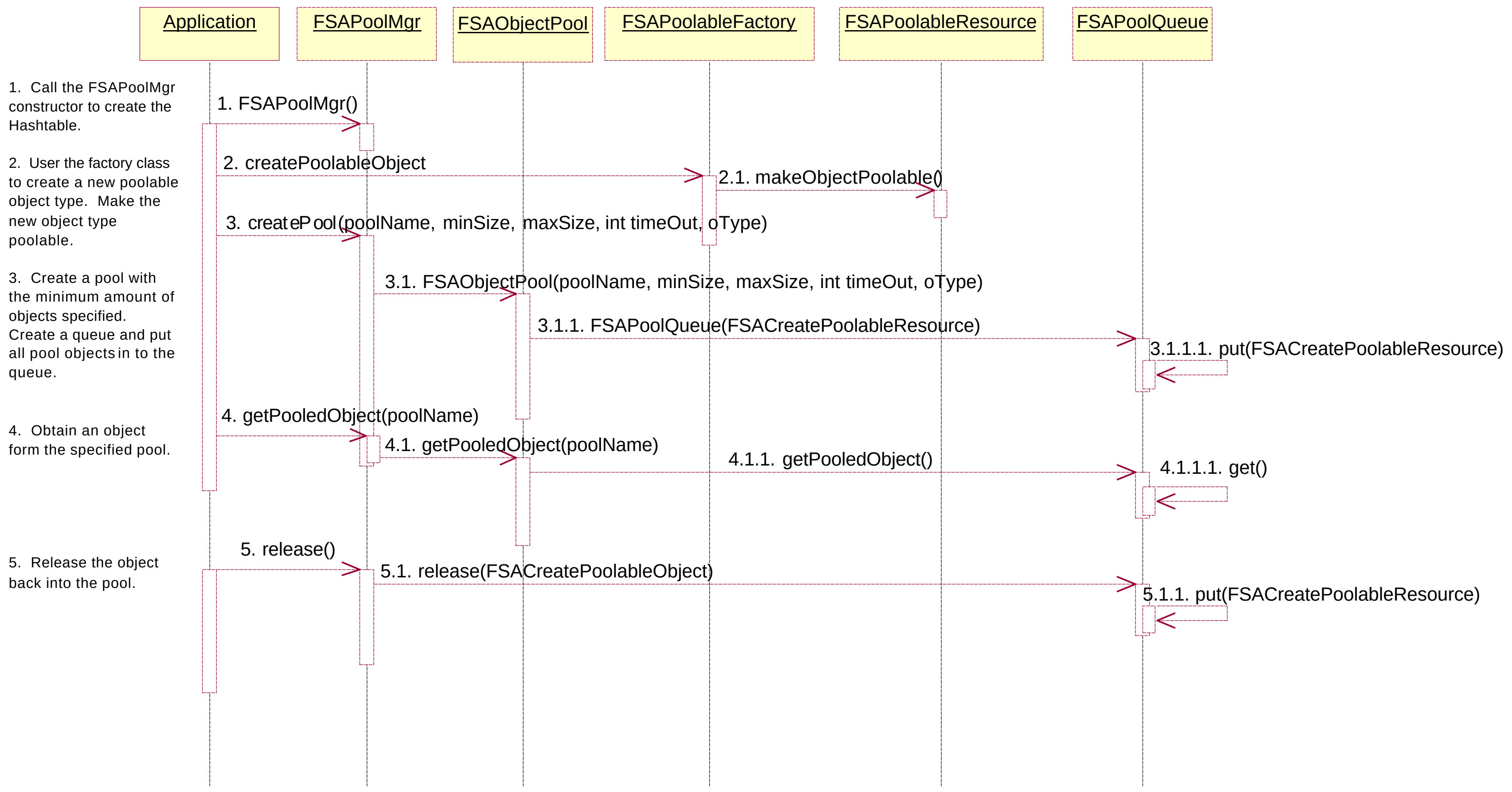
4.2.6 Class Diagram

4.2.6.1 Pooling Framework Class Diagram



4.2.7 Sequence Diagram

4.2.7.1 Pooling Framework Sequence Diagram


Application
FSAPoolMgr
FSAObjectPool
FSAPoolableFactory
FSAPoolableResource
FSAPoolQueue
1. FSAPoolMgr()
2. createPoolableObject
2.1. makeObjectPoolable()
3. createPool(poolName, minSize, maxSize, int timeOut, oType)
3.1. FSAObjectPool(poolName, minSize, maxSize, int timeOut, oType)
3.1.1. FSAPoolQueue(FSACreatePoolableResource)
3.1.1.1. put(FSACreatePoolableResource)
4. getPooledObject(poolName)
4.1. getPooledObject(poolName)
4.1.1. getPooledObject()
4.1.1.1. get()
5. release()
5.1. release(FSACreatePoolableObject)
5.1.1. put(FSACreatePoolableResource)

1. Call the FSAPoolMgr constructor to create the Hashtable.

2. User the factory class to create a new poolable object type. Make the new object type poolable.

3. Create a pool with the minimum amount of objects specified. Create a queue and put all pool objects in to the queue.

4. Obtain an object form the specified pool.

5. Release the object back into the pool.

4.2.8 References

- "Advanced Programming for the Java 2 Platform, " Chapter 8 continued: Connection Pooling

  http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/conpool.html

- "Build your own ObjectPool in Java to Boost Application Speed," Thomas E. Davis

  http://www.javaworld.com/javaworld/jw-06-1998/jw-06-object-pool_p.html

- "Dive into Connection Pooling with J2EE," Siva Visveswaran

  http://www.javaworld.com/javaworld/jw-10-2000/jw-1027-pool.html

- "Improve the Robustness and Performance of Your ObjectPool," Thomas E. Davis

  http://www.javaworld.com/javaworld/jw-08-1998/jw-08-object-pool_p.html

- Java Tip 67: Lazy instantiation by Philip Bishop and Nigel Warrant

  http://www.javaworld.com/javaworld/javatips/jw-javatip67_p.html

- Java Tip 78: Recycle broken objects in resource pools

  http://www.javaworld.com/javaworld/javatips/jw-javatip78_p.html

- "The Java Tutorial," Lesson - Threads: Doing Two or More Tasks at Once

  http://java.sun.com/docs/books/tutorial/essential/threads/index.html

- "Pooling Arbitrary Objects," researched by Ramchandar Varadarajan

  http://developer.java.sun.com/developer/qow/archive/138/index.jsp

- "Reduce, Reuse and Recycle: Reusing Objects - Part I," Angus Muir and Roman Bialach

  http://www.microjava.com/articles/techtalk/recycle

4.3 Session Framework

4.3.1 Purpose

The purpose of the ITA session framework is to provide a standard to simplify, and extend the implementation of user session / context information within the J2EE standard. The session framework will provide a common way to access user session / context information regardless of the storage method. The framework will decouple session information from the request, session, and cookie contexts; the framework will also wrap WebSphere session extension classes.

4.3.2 System Overview

The session framework provides a mechanism for the retrieval and manipulation of user session and context data stored in cookies, web server session variables, or in a data store. The framework also creates one common interface for application developers to access the session via any of these methods.

The session framework will provide context management service that stores a user's temporary data during their HTTP session. The session framework will provide for the following services on both the client-side and server-side contexts:

- A common interface to all HTTP variables (request, cookie, session)
- Storing temporary data that should persist across each web page presented to the user

The FSAContextManager class is the main class that application developers will call to obtain the session information. This class will require the calling application to provide the HttpServletRequest object and the type of method (client or server) to use to store session information.

The FSACookieRetrieval and FSASessionRetrieval are internal classes and are called by the ContextManager to access and manipulate the client and server session data.

4.3.3 Design Considerations

4.3.3.1 Assumptions and Dependencies

The session framework will function in a J2EE application server environment. As the current production servers for FSA are running IBM's WAS v. 3.5.3 and v. 3.5.5, the framework will be compiled using its required JDK version 1.2.2. It will also work with the current JavaServer Pages (1.1), Java Servlet (2.2), and Java Database Connectivity (2.0) specifications for this server. It will be built and tested on the Sun Solaris 2.6 and HP-UX

11.0 operating systems. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.3.3.2 Goals and Guidelines

The goal of the session framework is to provide a simple yet robust framework that could easily be utilized by any FSA development team using Java. The session framework provides the developer with a single interface to the session context regardless of where the session data is stored. This avoids duplicate development efforts by each application developer for every interaction with the session context. This framework will provide developers with the methods to use to access session data and prevent the developers from using deprecated APIs.

Cookies and URL rewriting should be enabled in Session Manager regardless of the context component used. It is important to note that even if the application developer chooses to store session data on the server, cookies or URL rewriting will still have to be used to store the session ID on the client. Additional guidelines for configuring the WebSphere Session Manager for the server-side context component can be found in section 4.3.9 of this document.

4.3.3.3 Development Methods

This framework will be developed using general object-oriented software development techniques. The standard class and sequence diagrams are provided in this document. These diagrams should assist developers who are unfamiliar with this framework.

4.3.4 System Architecture

4.3.4.1 Overview

The session framework provides the following session context storage methods:

- Client Side State Context Component: Small amounts of session data can be stored in cookies on the users' browsers
- Server Side Context Component:
  - Stateful In-Memory Context Component: Users' session data are stored in the server's memory in-between web requests using WebSphere session variables
  - Server Side Temporary Data Store Context Component: Users' session data is held in a temporary database or data store

4.3.4.2 Client-Side State Context Component

The client-side context component stores the users persistent state on the client by embedding the persistent state in a 'Cookie'. Cookies allow a web page to write a small amount of information to a user's browser in a safe and secure manner. This is a safe operation for the client and server. The browser will send the cookie back to the Web Server with each HTTP request.

Cookies are 'stateless'; no resources are used on the server in-between page requests. This makes the applications more scaleable and capable of handling a large number of users. Because the client stores its session information, the request does not have to be directed back to the same server in a clustered environment.

Cookies cannot be used if the users' browser does not support them or if the user disables the cookies function. Since the information stored in the cookie is passed in the URL, it is possible for users to see what information is being stored in the cookie. Any sensitive information must be encrypted before being sent to the browser. Given that cookies are passed between the user's browser and the web server each time a request is sent, using cookies is not a practical option for storing large amount of data since doing so would reduce network performance.

4.3.4.3 Server-Side Context Component

The WebSphere Session Manager Graphical User Interface (GUI) can be used to configure how session data is stored on the server-side. By default, WebSphere is configured to store session objects in memory. Administrators have the option of enabling persistent session management, which instructs WebSphere to place session objects in a database. The administrator can enable persistence by changing a setting in the WebSphere Session Manager GUI and configuring the datasource to use.

The WebSphere servlet engine supports the Java Servlet specification containing the interface javax.servlet.http.HttpSession. From an application development perspective, servlet and JSP code do not interact directly with the Session Manager object. Rather, the Session Manager supports the HttpSession interface, which developers use for session management. Depending on how the administrator configured the WAS Session Manager to handle the server-side context component, the WAS SessionManager will access the session information in the server's memory to the database transparently.

IBM provides an extension API to the HttpSession interface that the FSASessionRetreival class in the session framework will utilize in order to set and retrieve session data. The session framework will utilize the IBMSession interface in the com.ibm.websphere.servlet.session package.

4.3.4.3.1 Stateful In-Memory Context Component (Local Sessions)

Stateful in-memory context component refers to session objects being stored in WebSphere session variables. The data stored here is not persistent and will not be maintained should the server unexpectedly fail. Local session information is not shared with other clustered machines and users can only obtain their session information if they return to the web server instance holding their session information on subsequent access to a web site.

4.3.4.3.2 Server Side Temporary Data Store Context Component (Persistence)

In order to store session data in a data store, the WebSphere Session Manager configuration has to be modified. This ensures that if an application server fails, the session data is not lost. Persistence storage also allows for different servers in the same WebSphere domain to share access to a session if the same data source is specified for each Session Manager. This way, the user's session will be available regardless of which application server is accessed. Storing session information in a data store could result in slower performance and require greater consideration into the amount of connections available and database backup and recovery.

4.3.4.4 Subsystem Architecture: FSAContextManager

The FSAContextManager class can be called by applications access data stored on the client or stored on the server.

4.3.4.5 Subsystem Architecture: Interface FSADataRetrieval

This is the Interface implemented by FSACookieRetrieval and FSASessionRetrieval.

4.3.4.6 Subsystem Architecture: FSACookieRetrieval

The FSACookieRetrieval class extends FSADataRetrieval and is used by FSASContextManager to access client cookies.

4.3.4.7 Subsystem Architecture: FSASessionRetrieval

FSAContextManager uses the FSASessionRetrieval class, which extends FSADataRetrieval, to access session data stored on the server.

4.3.4.8 Subsystem Architecture: Interface IBMSession

The IBMSession interface is an IBM extension that extends and adds to the Java Servlet API. This interface extends the HttpSession for session support and increases web administrators' control in a clustered environment. Application developers should not call this class directly, but instead use the method provided in the FSASessionRetrieval class.

4.3.5 Detailed System Design

4.3.5.1 Component Definitions

4.3.5.1.1 FSAContextManager

Class Name:	FSAContextManager
Component:	Session
Description:	Public class
Package:	gov.ed.fsa.ita.session
Superclass:	Object

Attribute	**Type**	**Description**
Private:		
m_data	FSADataRetrieval	Stores the type of FSADataRetrieval object to access.
m_valueHash	Hashtable	The hashtable of key / value pairs.

Con/Destructors	**Arguments** (Type, Name)	**Description**
FSAContextManager	javax.servlet.HttpServletRequest request, Boolean manUpdate	Default Constructor for storing session data on the server. The Boolean value is set to true when manual update will be used.
FSAContextManager	javax.servlet.HttpServletRequest request, javax.servlet.HttpServletResponse response, String domain	Constructor used for storing session data on the client.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
getValue	String name	String	This method gets the value from the hashtable for the key.
setValue	String name, String value	void	This method sets the key and value in the hashtable.
deleteValue	String name	String	Clears the hashtable of all values for the specified key.
writeValues	none	Boolean	This method calls the appropriate sub-class method to put the information from the hashtable into the chosen storage type.
clearValues	none	void	Clears the hashtable of all key value pairs.
getKeys	none	java.util.Enumerati on	Returns an Enumeration of all keys in the hashtable.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
sync	none	void	Calls the IBMSession's manual update method - sync() to the database. When the FSASessionRetrieval constructor is called and the Boolean value passed in is true then application developer will have to ensure that sync is called at the end.

4.3.5.1.2 FSADataRetrieval

Interface Name:	Interface FSADataRetrieval
Component:	Session
Description:	Public interface that is implemented by FSACookieRetrieval and FSASessionRetrieval
Package:	gov.ed.fsa.ita.session
Superinterface:	None

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
GetValues	none	Hashtable	Returns a list of all key / value pairs for the request.
WriteValues	Hashtable hashtable	Boolean	Writes the hashtable to the proper client or server store.

4.3.5.1.3 FSACookieRetrieval (Protected)

Class Name:	FSACookieRetrieval
Component:	Session
Description:	This class will implement FSADataRetrieval and get and set the cookies on the client side.
Package:	gov.ed.fsa.ita.session
Superclass:	FSADataRetrieval

Attribute	**Type**	**Description**
Private:		
m_request	HttpServletRequest	The request object to set or obtain session information.
m_response	HttpServletResponse	Used to store the response object for the session.
m_path	String	Used to store a path name to the application to differentiate between sessions.

Con/Destructors	Arguments (Type, Name)	Description
FSACookieRetrieval	HttpServletRequest request, HttpServletResponse response, String domain	This is the default constructor.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
getValues	none	Hashtable	Get all key / values in the cookie and store in the hashtable.
writeValues	Hashtable hash	Boolean	Write the key / value pairs from the hashtable to new cookies. This method can only be called once during the life of the class since there is no method to delete cookies from responses. If the write command is called multiple times, a deleted value may not be removed and existing values will be added twice. Boolean returns true after the first time the method is called and returns false if the write method has been called more than once already.

4.3.5.1.4 FSASessionRetrieval (Protected)

Class Name:	FSASessionRetrieval
Component:	Session
Description:	This class will get and set the session variables.
Package:	gov.ed.fsa.ita.session
Superclass:	FSADataRetrieval

Attribute	Type	Description
Private:		
m_request	javax.servlet.HttpServletRequest	The request object to set or obtain session information.
m_session	HttpSession	The session obtained from the request.
m_update	Boolean	Default to false. Set to true if manual updating of the Session table will be used.
m_valueHash	java.util.Hashtable	Hashtable to store key value pairs.

Con/Destructors	Arguments (Type, Name)	Description
FSASessionRetrieval	javax.servlet.HttpServletRequest request, Boolean manualUpdate	This is the constructor to set if manually updating the Session table. Pass true for manualUpdate.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
cookieTest	javax.servlet.HttpSession session	Boolean	Tests whether the user's browser accepts cookies to store the session ID.
getValues	none	java.util.Hashtable	Get all key / values in the cookie and store in the hashtable.
writeValues	java.util.Hashtable hash	Boolean	Write the key / value pairs from the hashtable to the session. This method can be called more than once because the session values will be removed first before the hashtable values are written. Boolean will always return true.

4.3.5.1.5 IBMSession

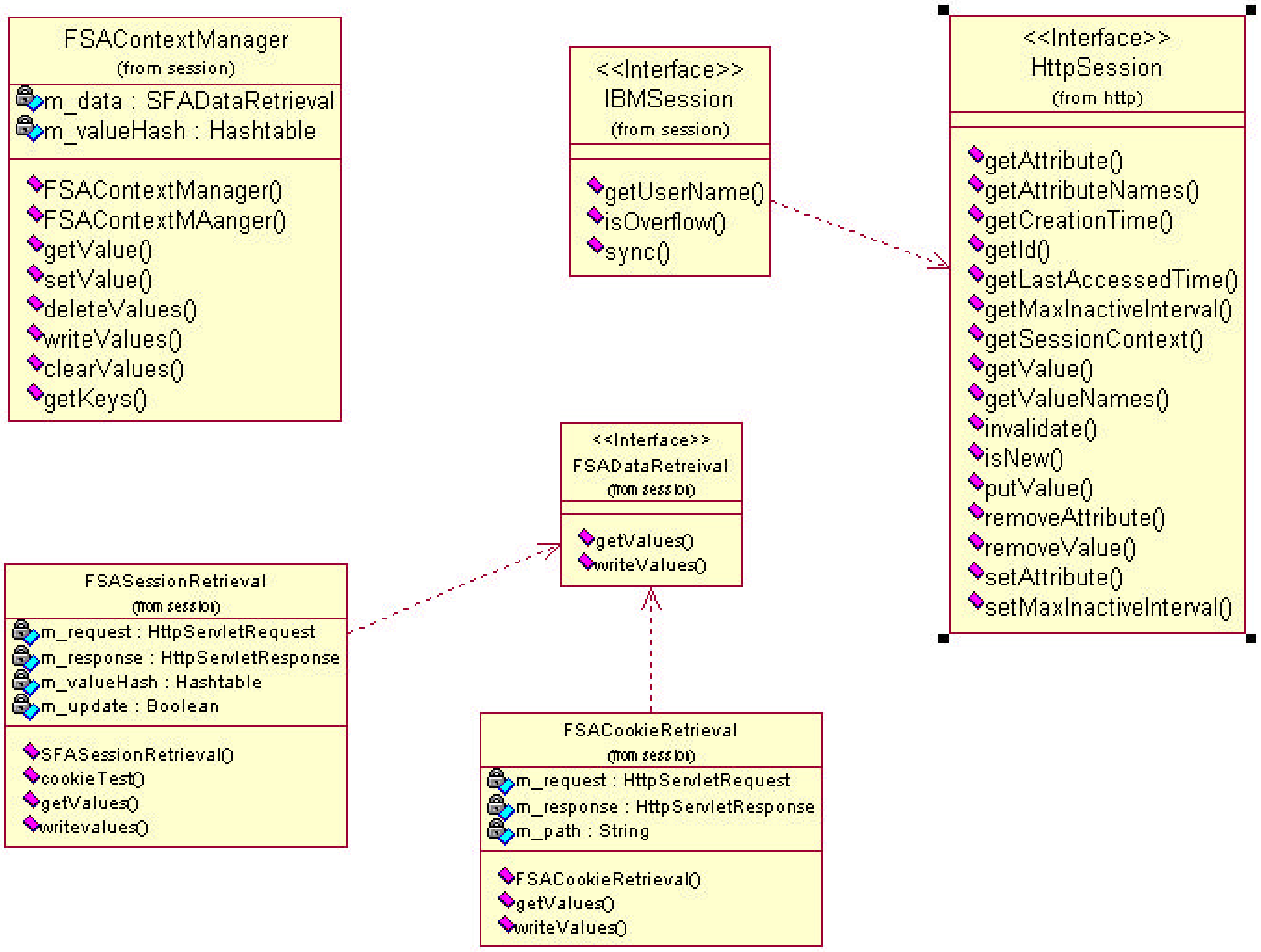
Interface Name	IBMSession
Component:	IBM Extension to the Servlet API 2.1
Description:	The IBMSession interface extends the HttpSession interface of the Servlet API to: • allow the session to be maintained in a clustered environment (via object serialization) • provide a measure of security for when a servlet attempts to access a session • allow customer control of the WebSphere concept of a HttpSession transaction in a clustered, database mode of operation.
Package:	com.ibm.websphere.servlet.session
Superinterface:	java.io.Externalizable

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
getUserName	none	String	Gets the username associated with the session.
isOverflow	none	Boolean	Determines whether Allow Overflow is enabled. Allow Overflow specifies whether to allow the number of sessions in memory to exceed the value specified by the Base In-memory Size property in the WebSphere Session Manager.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
sync	none	void	Used when WebSphere is set to manual mode to allow the application to decide when a session should be stored persistently.

4.3.6 Class Diagram

4.3.6.1 Session Framework Class Diagram



4.3.7 Sequence Diagram

4.3.7.1 Session Framework Sequence Diagram

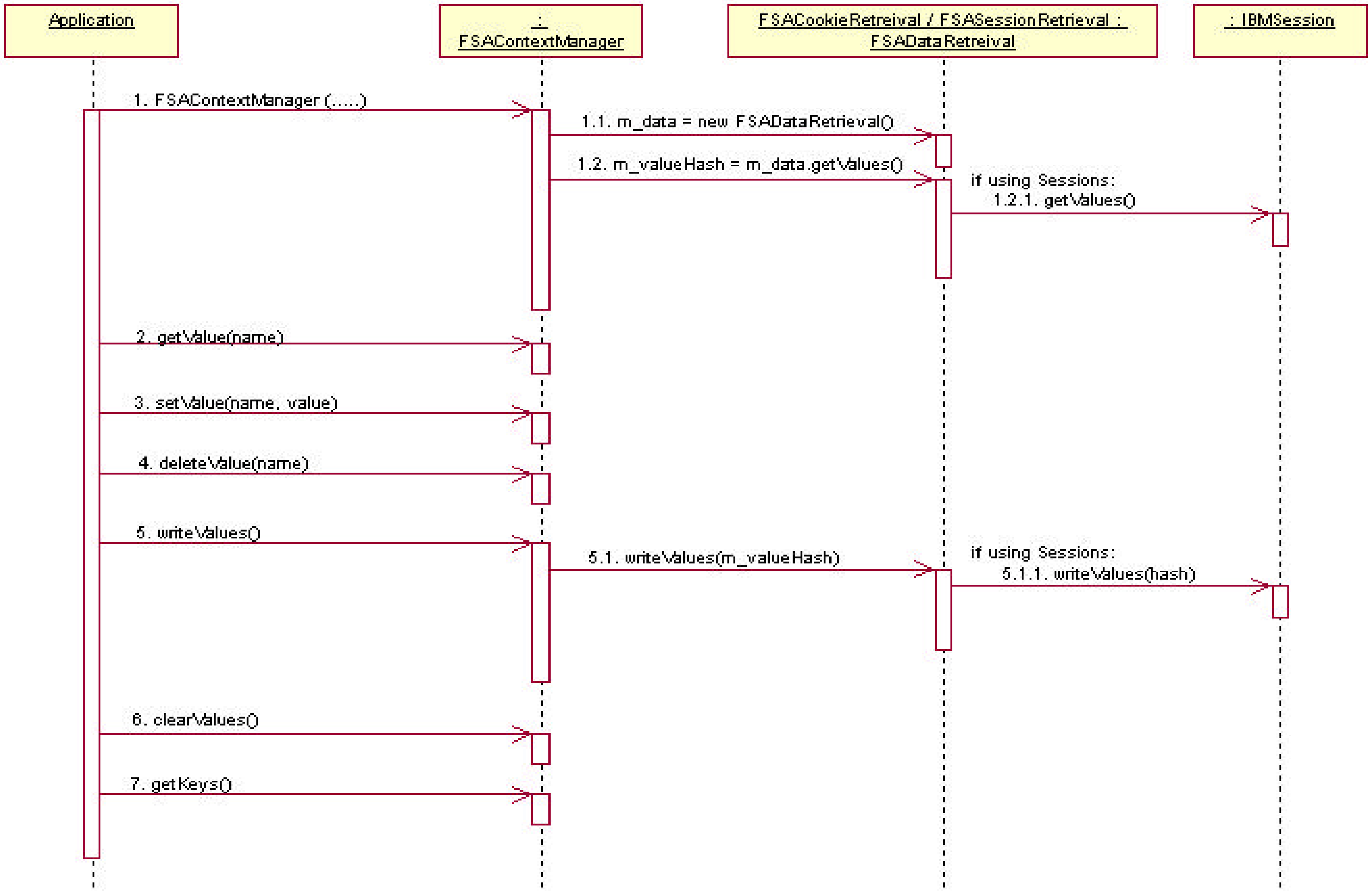
1. If using Sessions to store data: FSAContextManager (request, manuUpdate). If using Cookies to store data: FSA(ContextManager (request, response, path). The ContextManager will use the session or cookie retrieval methods to obtain any values that may already exist and populate it in a hash table.

2. The application has the ability to get, set, and delete any values stored in the ContextManager hash table.

5. The application can ask the ContextManager to write any data stored in the Hashtable to the storage medium. If using Sessions, writeValues will take the place of IBMSession's sync method.

6. clearValues is used to clear the Hashtable of any values.

7. getKeys returns an Enumeration of all key stored in the Hashtable.



4.3.8 Data Model

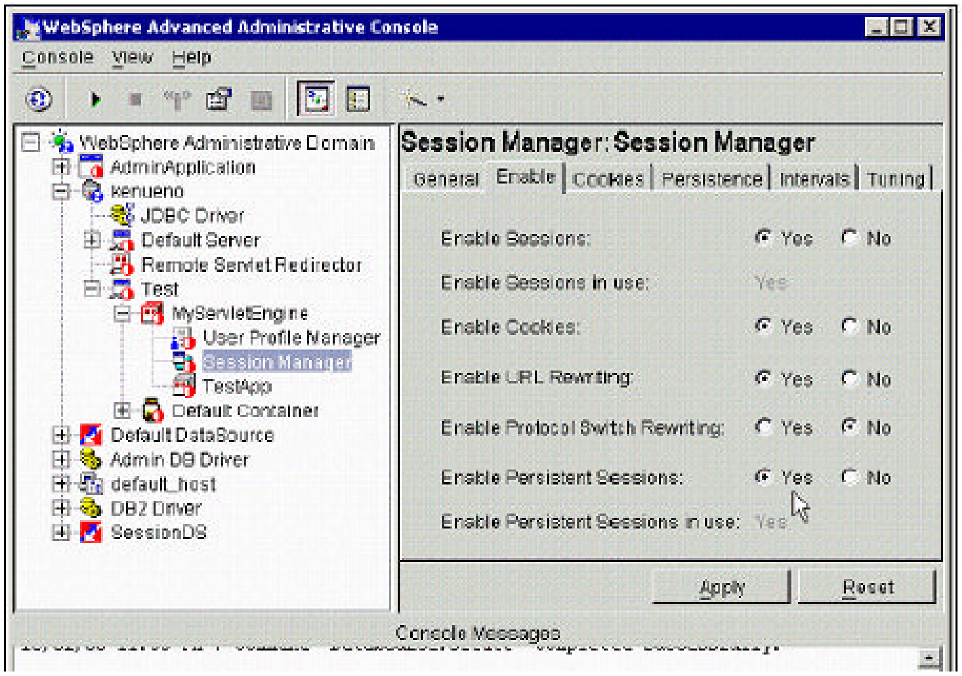
This table depicts the data model that will be used to store the persistence data stored in WebSphere database Sessions table.

Name	Null?	Type	Description
ID	NOT NULL	VARCHAR2(64)	Unique identifier
PROPID	NOT NULL	VARCHAR2(64)	
APPNAME		VARCHAR2(64)	Application name
LISTENERCNT		NUMBER(38)	
LASTACCESS		NUMBER(38)	Last time app. was accessed
CREATIONTIME		NUMBER(38)	Time of session creation
MAXINACTIVETIME		NUMBER(38)	Maximum idle time allowed
USERNAME		VARCHAR2(256)	User name
SMALL		RAW(2000)	Store data that is less than 3KB
MEDIUM		LONG RAW	Store data that is 3KB to 2MB
LARGE		RAW(1)	Store data that is greater than 2MB[1]

[1] This is an estimated number.

4.3.9 WebSphere Session Manager Configuration

The Session Manager property panes have to be configured for each application servlet engine according to the following guidelines in order for the application to utilize the FSAContextManager class. The guidelines described below are general configuration guidelines; please refer to the Best Practices for using HTTP Session document in the reference section for a complete description of each option.



Session Manager: Session Manager

Enable Sessions:	[✓ Yes] No
Enable Sessions in use:	Yes
Enable Cookies:	[✓ Yes] No
Enable URL Rewriting:	[✓ Yes] No
Enable Protocol Switch Rewriting:	Yes [✓ No]
Enable Persistent Sessions:	[✓ Yes] No
Enable Persistent Sessions in use:	Yes

Figure 4: Sample Session Manager properties configuration screen.

In the *Enable* tab of the Session Manager properties pane, 'Enable Sessions' and 'Enable Cookies' should have the radio button for the option 'Yes' darkened. By enabling both sessions and cookies, application developers will have the flexibility of storing session information on either the client or the server. 'Enable URL Rewriting' should also be set to 'Yes' so that users whose browsers do not accept cookies can still interact with applications that support URL rewriting. Government regulations also prohibit the use of cookies (in some cases) to store user information and URL rewriting would be needed then to maintain the session state.

If both cookies and URL rewriting are enabled and the application developer does not provide code to perform the URL rewriting then only cookies will be used. If URL rewriting is used, the cookie information will still exist as well.

If the application developer wants to store the session information on the server in a database, then 'Enable Persistent Sessions' should be enabled. Persistent sessions should be utilized for maintaining session affinity. WebSphere automatically performs the required steps necessary to store the data in session variables or in the database without developer intervention.

In the *Cookies* tab, the 'Cookie Name' option has to maintain the default value of 'sessionid' since it is a requirement of the J2EE specification. The *Persistence* tab has to be configured with the data source information if Persistent Sessions has been enabled. The 'Invalidate Time' option in the *Intervals* tab will be different depending on the needs of each application. The options to select in the *Tuning* tab will be different for each application.

In general, 'Using Cache' and 'Allow Overflow' should be enabled and the 'Base Memory Size' should be increased substantially from the default. It is considered to be a best practice to keep stored session data to less than 4KB, but if it is not possible then the 'Using Multirow Sessions' option should be examined more closely. Detailed description and facts to consider before selecting an option can be found in the Best Practices for using HTTP Session document.

Tab	Setting	Values
Enable	Enable Sessions	Yes
Enable	Enable Cookies	Yes
Enable	Enable URL Rewriting	Yes
Enable	Enable Persistent Sessions	Yes (if storing data and using clustering)
Cookies	Cookie Name	sesessionid
Persistence	**4.3.9.1 All Settings**	Include data source information if persistence is used
Intervals	Invalidate Time	Depends on the application requirements
Tuning	**4.3.9.2 All Settings**	Depends on the application requirements
Tuning	Use Cache	Generally Enabled
	Allow Overflow	Generally Enabled
	Base Memory Size	Generally be increased substantially from the default

4.3.10 References

- Best Practices for Session Programming: WebSphere Application Server

  http://www-4.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/0404010108.html

- Maintaining Session Data with the WebSphere Session Manager

  http://www6.software.ibm.com/devtools/news0801/art26.htm

- Session Manager Properties

  http://www-4.ibm.com/software/webservers/appserv/doc/v35/ae/infocenter/was/06061100.html

- WebSphere Application Server Best Practices using HTTP Sessions

  http://www.106.ibm.com/developerworks/patterns/guidelines/HTTP_Session_Best_Practice.pdf

4.4 FTP Framework

4.4.1 Purpose

The purpose of the ITA FTP framework is to provide a standard interface to applications for FTP commands return codes. The framework allows applications to create FTP connections and execute a variety of FTP commands (which include the upload and download of files). A graphical user interface allows applications to easily add FTP functionality.

4.4.2 System Overview

The FTP framework is created using the Java programming language. The FTP framework will run on IBM's WebSphere Application Server (WAS). The FTP framework provides a mechanism to create FTP connections and execute FTP commands. The framework also allows the application to choose the level of security (SSL) to be used for the information transfer. The framework creates one common application-programming interface (API) for application developers. A graphical user interface is also available for applications to use.

4.4.2.1 What is FTP?

File Transfer Protocol is a client-server protocol that allows a user on one computer to transfer files to and from another computer over a TCP/IP network. FTP utilizes two ports, a 'data' port and a ‘command’ port (also known as the control port). The command port is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands. The data port is used to transfer data (directory listings, files, etc) between the client and the server.

FTP has two connection modes called active mode and passive mode. Both modes use the command port and data port. However, the determination of the ports (client or server) is different. In active mode, the client initiates the command port to the server and the server initiates the data port back to the client. In passive mode, the client initiates both the command port and data port to the server.

4.4.2.1.1 Active Mode

The following are the steps performed during active mode:

- The client connects from a random unprivileged port (n > 1024) to the server's command port
- The client sends the server a port number (n + 1) to the server (via the PORT command)
- The server connects to the client's specified data port from its local data port

However, there is a problem with active mode when there is a client firewall. In active mode, the FTP client does not make the actual connection to the data port of the server. The client only tells the server what port it is listening on and the server connects back to the specified port on the client. To the firewall, the server connection appears to be an outside system initiating a connection to an internal client and the firewall will not always allow the connection. In this case, passive mode should be used.

4.4.2.1.2 Passive Mode

The following are the steps performed during passive mode:

- The client opens two random unprivileged ports (n > 1024 and n+1)
- The client issues a PASV command on the first port
- The server opens a random unprivileged port and sends the port number to the client (via the PORT command)
- The client connects to the server's specified data port from second opened port

Passive mode solves the problem of having a firewall on the client side. However, it does pose other issues. The biggest issue is the need to allow any remote connection to high numbered ports on the server. However, most FTP daemons allow the specification of a range of ports for the use of FTP.

4.4.2.2 Java Secure Socket Extension

The Java Secure Socket Extension (JSSE) 1.0.2 is a set of Java packages that enable secure Internet communications. The JSSE implements a Java version of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols and includes functionality for data encryption, server authentication, and message integrity. Using JSSE, developers can provide for the secure passage of data between a client and a server running FTP over TCP/IP.

The FTP framework utilizes the javax.net.ssl package. This package contains classes that allow applications to create secure sockets for use during the FTP connection process.

4.4.3 Design Considerations

4.4.3.1 Assumptions and Dependencies

The FTP framework will function in a J2EE application server environment. As the current production server for FSA is IBM's WAS v. 3.5, the framework will be compiled using its required JDK version 1.2.2. It will also work with the current JavaServer Pages (1.1) and Java Servlet (2.2) specifications for this server. It will be fully tested on both the Sun Solaris 2.6 and the HP-UX 11.0 operating systems. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.4.3.2 Goals and Guidelines

The goal of the FTP framework is to provide a simple and robust framework that may be applied by FSA application teams developing in the Java environment, as well as WebSphere. The FTP framework abstracts the syntax of FTP commands into the architecture layer. This allows applications to execute FTP commands without having to consider command syntax or command return codes. The graphical user interface allows applications to quickly insert FTP functionality into their own front end.

4.4.3.3 Development Methods

This framework will be developed using general object-oriented software development techniques. The standard class and sequence diagrams are provided in this document. These diagrams are intended to assist developers who are unfamiliar with this framework.

4.4.4 System Architecture

4.4.4.1 Overview

The FTP framework allows an application to make FTP connections and execute FTP commands without having to manage command syntax and command return codes. There are Java classes that create the FTP framework, and there are the JSP files and Java classes that create the graphical user interface to the FTP APIs. All of these parts are discussed in-depth in the detailed design section.

4.4.4.2 Subsystem Architecture for FTP Framework

4.4.4.2.1 Subsystem Architecture: FSAFtpClient

The FSAFtpClient class is the wrapper class for all FTP commands.

4.4.4.2.2 Subsystem Architecture: FSAReplyCodes

Reply codes for debugging purposes. The codes' English interpretations are stored in a hashtable for lookup. They are taken directly from RFC959 – the link for which can be found in the reference section.

4.4.4.2.3 Subsystem Architecture: FSAFtpControlSocket

The FSAFtpControlSocket class supports client-side FTP command operations.

4.4.4.2.4 Subsystem Architecture: FSAFtpDataSocket

The FSAFtpDataSocket class Supports client-side FTP DataSocket in Passive and Active Mode. It is a wrapper for Socket and ServerSocket.

4.4.4.2.5 Subsystem Architecture: FSAFtpException

The FSAFtpException class is for FTP specific exceptions.

4.4.4.3 Subsystem Architecture for Graphical User Interface

4.4.4.3.1 Overview

The graphical user interface for the FTP framework uses the web conversation framework. For more information on the web conversation framework, see the appropriate section in this document.

4.4.4.3.2 Subsystem Architecture: FTPConnectAction

The FTPConnectAction class assists in the display of an FTP connection.

4.4.4.3.3 Subsystem Architecture: FTPConnectForm

The FTPConnectForm class holds the data to be displayed.

4.4.4.3.4 Subsystem Architecture: FTPMoveFileAction

The FTPMoveFileAction class assists in the display.

4.4.4.3.5 Subsystem Architecture: FTPMoveFileForm

The FTPMoveFileForm class holds the data to be displayed.

4.4.5 Detailed System Design

4.4.5.1 Component Definitions

4.4.5.1.1 FSAFtpClient

Class Name:	FSAFtpClient
Component:	FTP
Description:	This class is the wrapper class for all FTP commands.
Package:	gov.ed.fsa.ita.ftp
Superclass:	Object

Attribute	Type	Description
Public:		
m_host	static String	The host string to make the FTP connection
m_username	static String	The username to make the connection
m_password	static String	The password to make the connection
m_port	static String	The default port number

Con/Destructors	Arguments (Type, Name)	Description
FSAFtpClient	None	Default Constructor
FSAFtpClient	String host, boolean isSecure	Constructor that also creates a control port to the specified host.
FSAFtpClient	String host, String port, boolean isSecure	Constructor that also creates a control port to the specified host and port.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
connect	String host	boolean	This method creates an FTP connection to the host passed in. Uses the default port number.
connect	String host, String port	boolean	This method creates an FTP connection to the host and port passed in.
login	String username, String password	void	This method logs into an account on the FTP server.
setUser	String username	void	This method identifies the user to the FTP server.
setPassword	String password	void	This method identifies the password to the FTP server.
changeDirectory	String pathname	void	This method changes the remote working directory.
quit	None	void	This method quits the FTP session.
setTransferMode	String transferMode	void	This method sets the data transfer mode. (ASCII or binary)
putFile	String filename	void	This method puts the file from the client to the server
getFile	String filename	void	This method gets the file from the server to the client.

4.4.5.1.2 FSAReplyCodes

Class Name:	FSAReplyCodes
Component:	FTP
Description:	This class contains all the possible reply codes according to RFC959. It also contains a static initializer to populate the hashtable with all the reply codes and their descriptions.
Package:	gov.ed.fsaita.ftp
Superclass:	Hashtable

Attribute	Type	Description
Private:		
replyCodes	static Hashtable	The hashtable that stores the reply codes.
Public:		

Attribute	Type	Description
POSITIVE_PRELIMINARY	static final int	First-digit encoding of reply values. The value is set to 1.
POSITIVE_COMPLETION	static final int	First-digit encoding of reply values. The value is set to 2.
POSITIVE_INTERMEDIATE	static final int	First-digit encoding of reply values. The value is set to 3.
TRANSIENT_NEGATIVE_COMPLETION	static final int	First-digit encoding of reply values. The value is set to 4.
PERMANENT_NEGATIVE_COMPLETION	static final int	First-digit encoding of reply values. The value is set to 5.
SYNTAX	static final int	Second-digit encoding of reply values. The value is set to 0.
INFORMATION	static final int	Second-digit encoding of reply values. The value is set to 1.
CONNECTIONS	static final int	Second-digit encoding of reply values. The value is set to 2.
AUTHENTICATION_AND_ACCOUNTING	static final int	Second-digit encoding of reply values. The value is set to 3.
UNSPECIFIED	static final int	Second-digit encoding of reply values. The value is set to 4.
FILE_SYSTEM	static final int	Second-digit encoding of reply values. The value is set to 5.
RESTART_MARKER_REPLY	static final int	100 block of reply codes. The value is set to 110.
SERVICE_READY_IN_NNN_MINUTES	static final int	100 block of reply codes. The value is set to 120.
DATA_CONNECTION_ALREADY_OPEN_TRANFER_STARTING	static final int	100 block of reply codes. The value is set to 125.
FILE_STATUS_OK_ABOUT_TO_OPEN_DATA_CONNECTION	static final int	100 block of reply codes. The value is set to 150.
COMMAND_OK = 200;	static final int	200 block of reply codes. The value is set to 200.
COMMAND_NOT_IMPLEMENTED_SUPERFLUOUS	static final int	200 block of reply codes. The value is set to 202.
STATUS_OR_HELP_REPLY	static final int	200 block of reply codes. The value is set to 211.
DIRECTORY_STATUS	static final int	200 block of reply codes. The value is set to 212.
FILE_STATUS	static final int	200 block of reply codes. The value is set to 213.
HELP_MESSAGE	static final int	200 block of reply codes. The value is set to 214.
SYSTEM_TYPE	static final int	200 block of reply codes. The value is set to 215.
SERVICE_READY_FOR_NEW_USER	static final int	200 block of reply codes. The value is set to 220.
SERVICE_CLOSING_CONTROL_CONNECTION	static final int	200 block of reply codes. The value is set to 221.
DATA_CONNECTION_OPEN_NO_TRANSFER_IN_PROGRESS	static final int	200 block of reply codes. The value is set to 225.
CLOSING_DATA_CONNECTION_AFTER_SUCCESSFUL_ACTION	static final int	200 block of reply codes. The value is set to 226.

Attribute	Type	Description
ENTERING_PASSIVE_MODE	static final int	200 block of reply codes. The value is set to 227.
USER_LOGGED_IN	static final int	200 block of reply codes. The value is set to 230.
FILE_ACTION_OK_AND_COMPLETED	static final int	200 block of reply codes. The value is set to 250.
DIRECTORY_CREATED	static final int	200 block of reply codes. The value is set to 257.
USER_NAME_OK	static final int	300 block of reply codes. The value is set to 331.
NEED_ACCOUNT_FOR_LOGIN	static final int	300 block of reply codes. The value is set to 332.
`REQUESTED_ACTION_PENDING_MORE_INFO	static final int	300 block of reply codes. The value is set to 350.
SERVICE_NOT_AVAILABLE_CLOSING_CONTROL_CONNECTION	static final int	400 block of reply codes. The value is set to 421.
CANNOT_OPEN_DATA_CONNECTION	static final int	400 block of reply codes. The value is set to 425.
CONNECTION_CLOSED_TRANSFER_ABORTED	static final int	400 block of reply codes. The value is set to 426.
ACTION_NOT_TAKEN_FILE_UNAVAILABLE_FILE_BUSY	static final int	400 block of reply codes. The value is set to 450.
ACTION_ABORTED_LOCAL_ERROR	static final int	400 block of reply codes. The value is set to 451.
ACTION_NOT_TAKEN_INSUFFICIENT_STORAGE_SPACE	static final int	400 block of reply codes. The value is set to 452.
SYNTAX_ERROR_IN_COMMAND	static final int	500 block of reply codes. The value is set to 500.
SYNTAX_ERROR_IN_PARAMETERS	static final int	500 block of reply codes. The value is set to 501.
COMMAND_NOT_IMPLEMENTED	static final int	500 block of reply codes. The value is set to 502.
BAD_SEQUENCE_OF_COMMANDS	static final int	500 block of reply codes. The value is set to 503.
COMMAND_NOT_IMPLEMENTED_FOR_THAT_PARAMETER	static final int	500 block of reply codes. The value is set to 504.
NOT_LOGGED_IN	static final int	500 block of reply codes. The value is set to 530.
NEED_ACCOUNT	static final int	500 block of reply codes. The value is set to 532.
ACTION_NOT_TAKEN_FILE_UNAVAILABLE	static final int	500 block of reply codes. The value is set to 550.
ACTION_ABORTED_PAGE_TYPE_UNKNOWN	static final int	500 block of reply codes. The value is set to 551.
ACTION_ABORTED_EXCEEDED_STORAGE_ALLOCATION	static final int	500 block of reply codes. The value is set to 552.
ACTION_NOT_TAKEN_FILE_NAME_NOT_ALLOWED	static final int	500 block of reply codes. The value is set to 553.

Con/Destructors	Arguments (Type, Name)	Description
FSAReplyCodes	None	Default Constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
changeCode	int code	String	Converts the numeric FTP code specified in the parameter code to an English description, taken from RFC959.

4.4.5.1.3 FSAFtpControlSocket

Class Name:	FSAFtpControlSocket
Component:	FTP
Description:	This class performs all the operations having to do with the control socket of an FTP connection.
Package:	gov.ed. fsa.ita.ftp
Superclass:	Object

Attribute	Type	Description

Con/Destructors	Arguments (Type, Name)	Description
FSAFtpControlSocket	None	Default Constructor
FSAFtpControlSocket	String host, boolean isSecure	Constructor that also creates a control port to the specified host.
FSAFtpControlSocket	String host, String port, boolean isSecure	Constructor that also creates a control port to the specified host and port.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
initStreams	None	void	Obtain the reader / writer streams for this connection.
readReply	None	String	Read the FTP server's reply to a previously issued command.
validateReply	String replyCode, String acceptedReplyCode	void	Validate the response the host has supplied against the expected reply. If an unexpected reply is received, an exception is thrown, setting the message to that returned by the FTP server.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
validateReply	String replyCode, String [] acceptedReplyCodes	void	Validate the response the host has supplied against the expected reply. If an unexpected reply is received, an exception is thrown, setting the message to that returned by the FTP server.
Public:			
getRemoteHostName	None	String	Get the name of the remote host.
setTimeout	int millis	void	Set the TCP timeout on the underlying control socket.
logout	None	void	Quit this FTP session and clean up.
createDataSocket	String connectMode	FSAFtpDataSocket	Request a data socket be created on the server, connect to it and return our connected socket.
sendCommand	String command	String	Send a command to the FTP server and return the server's reply.

4.4.5.1.4 FSAFtpDataSocket

Class Name:	FSAFtpDataSocket
Component:	FTP
Description:	This class performs all the operations having to do with the data socket of an FTP connection.
Package:	gov.ed. fsa.ita.ftp
Superclass:	Object

Attribute	Type	Description
passiveSocket	ServerSocket	The underlying socket for PASV connection or Socket accepted from server.
activeSocket	Socket	The underlying socket for Active connection.

Con/Destructors	Arguments (Type, Name)	Description
FSAFtpDataSocket	None	Default Constructor
FSAFtpDataSocket	Socket socket, boolean isSecure	Constructor that creates a data socket connected to an active socket.
FSAFtpDataSocket	ServerSocket socket, boolean isSecure	Constructor that creates a data socket connected to an active socket.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
close	None	void	This method closes the socket.
setTimeout	int millis	void	Set the TCP timeout on the underlying control socket.
getOutputStream	None	OutputStream	Gets the output stream of the connection.
getInputStream	None	InputStream	Gets the input stream of the connection.

4.4.5.1.5 FSAFtpException

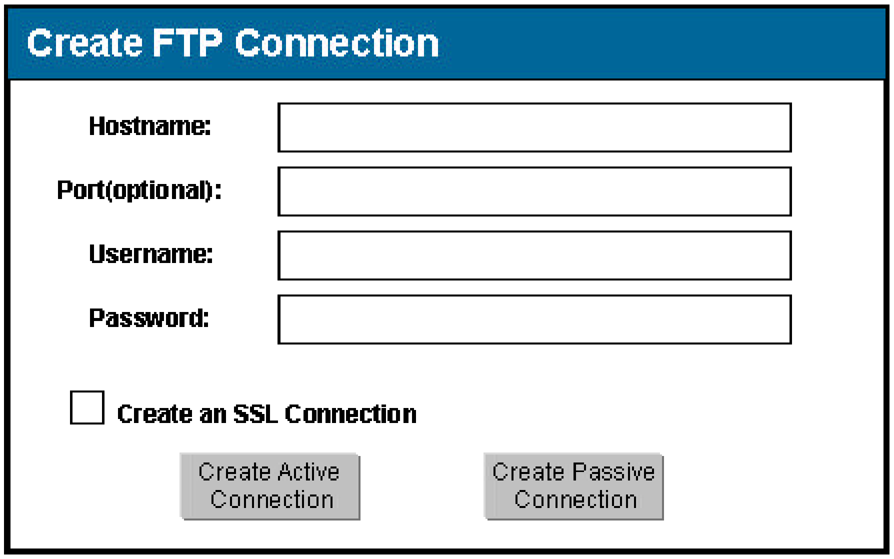
Class Name:	FSAFtpException
Component:	FTP
Description:	This class handles exceptions within the FTP package.
Package:	gov.ed. fsa.ita.ftp
Superclass:	FSAException

Attribute	Type	Description

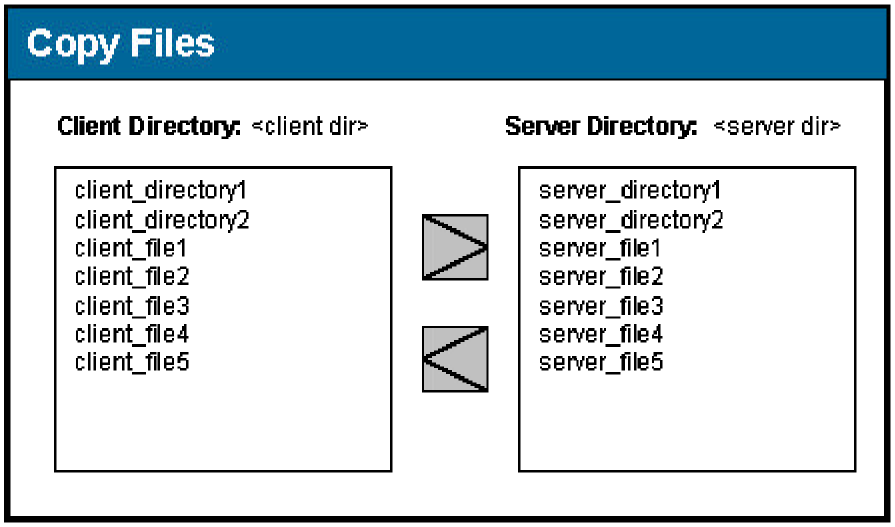
Con/Destructors	Arguments (Type, Name)	Description
FSAFtpException	String msg	Default Constructor
FSAFtpException	String msg String replyCode	Constructor that allows for a reply code.

Graphical User Interface Screenshots

4.4.5.1.6 To make a FTP connection – FTPConnection.jsp

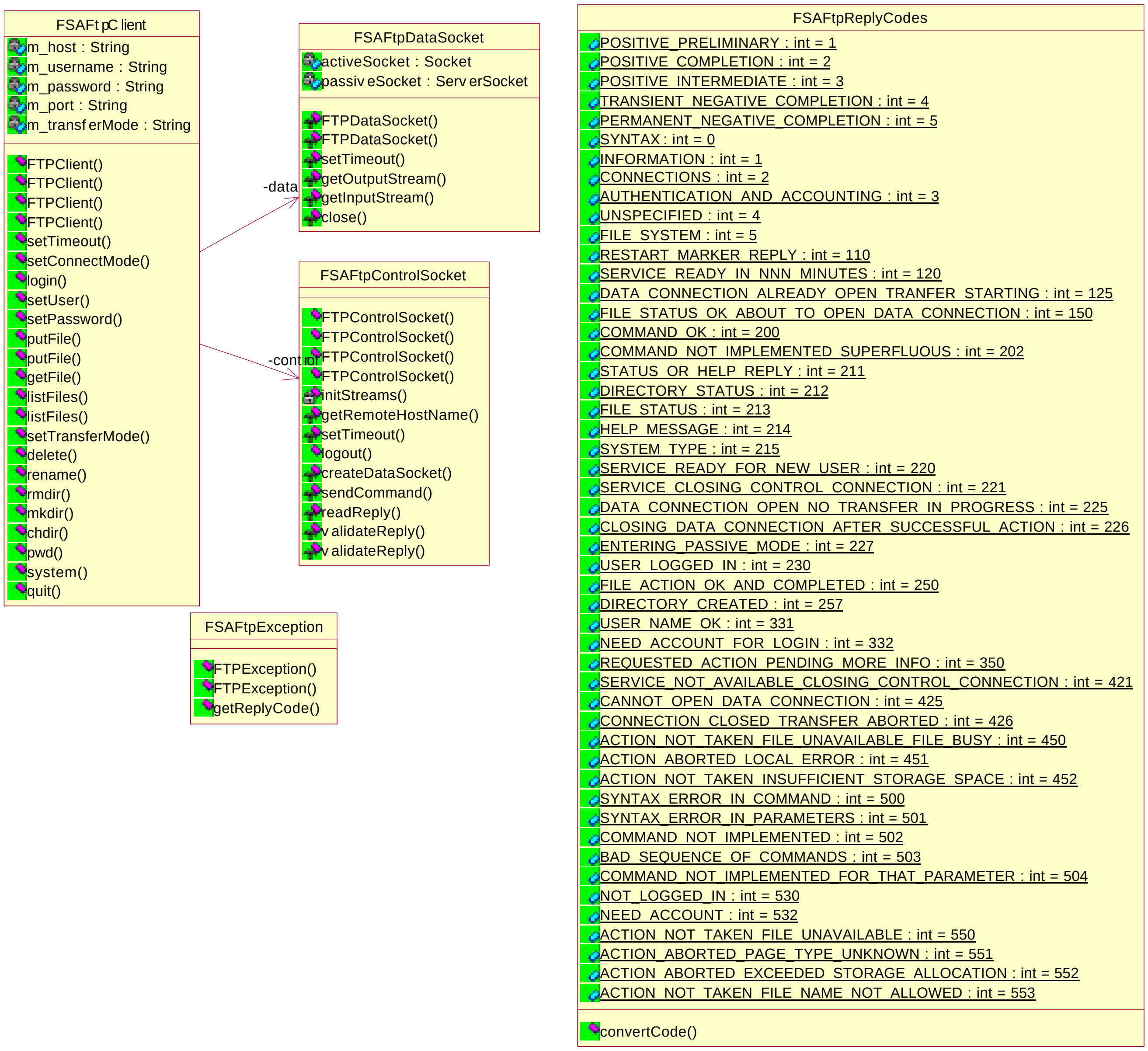


4.4.5.1.7 To move files between the client and the server – FTPMoveFile.jsp



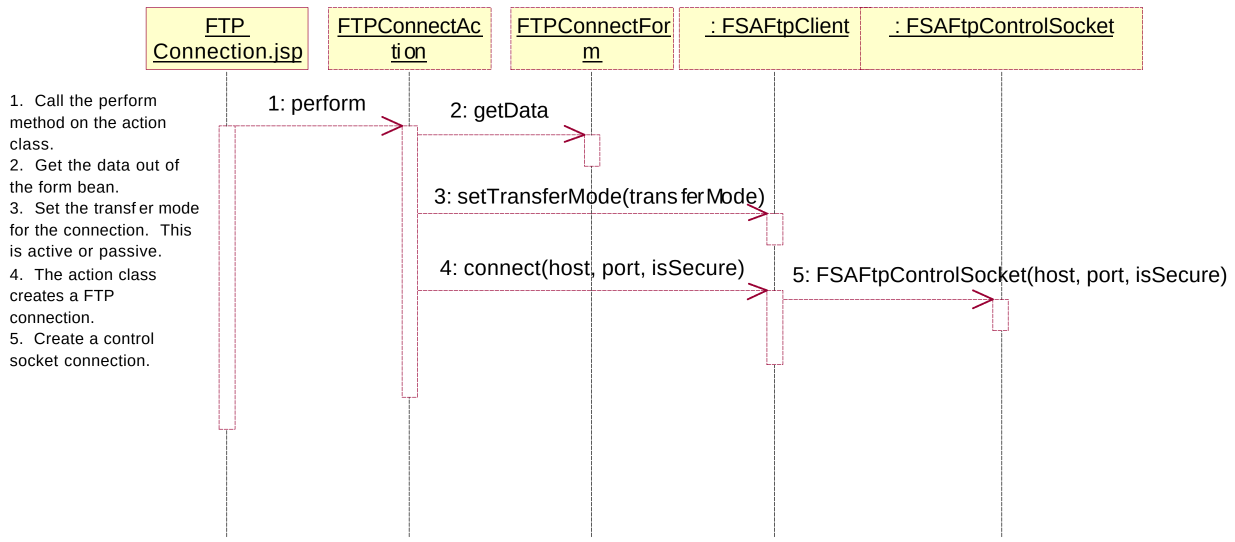
4.4.6 Class Diagram

4.4.6.1 FTP Classes


FSAFtpClient
m_host : String
m_username : String
m_password : String
m_port : String
m_transferMode : String
FTPClient()
FTPClient()
FTPClient()
FTPClient()
setTimeout()
setConnectMode()
login()
setUser()
setPassword()
putFile()
putFile()
getFile()
listFiles()
listFiles()
setTransferMode()
delete()
rename()
rmdir()
mkdir()
chdir()
pwd()
system()
quit()
-data
-control
FSAFtpDataSocket
activeSocket : Socket
passiveSocket : ServerSocket
FTPDataSocket()
FTPDataSocket()
setTimeout()
getOutputStream()
getInputStream()
close()
FSAFtpControlSocket
FTPControlSocket()
FTPControlSocket()
FTPControlSocket()
FTPControlSocket()
initStreams()
getRemoteHostName()
setTimeout()
logout()
createDataSocket()
sendCommand()
readReply()
validateReply()
validateReply()
FSAFtpException
FTPException()
FTPException()
getReplyCode()
FSAFtpReplyCodes
POSITIVE_PRELIMINARY : int = 1
POSITIVE_COMPLETION : int = 2
POSITIVE_INTERMEDIATE : int = 3
TRANSIENT_NEGATIVE_COMPLETION : int = 4
PERMANENT_NEGATIVE_COMPLETION : int = 5
SYNTAX : int = 0
INFORMATION : int = 1
CONNECTIONS : int = 2
AUTHENTICATION_AND_ACCOUNTING : int = 3
UNSPECIFIED : int = 4
FILE_SYSTEM : int = 5
RESTART_MARKER_REPLY : int = 110
SERVICE_READY_IN_NNN_MINUTES : int = 120
DATA_CONNECTION_ALREADY_OPEN_TRANFER_STARTING : int = 125
FILE_STATUS_OK_ABOUT_TO_OPEN_DATA_CONNECTION : int = 150
COMMAND_OK : int = 200
COMMAND_NOT_IMPLEMENTED_SUPERFLUOUS : int = 202
STATUS_OR_HELP_REPLY : int = 211
DIRECTORY_STATUS : int = 212
FILE_STATUS : int = 213
HELP_MESSAGE : int = 214
SYSTEM_TYPE : int = 215
SERVICE_READY_FOR_NEW_USER : int = 220
SERVICE_CLOSING_CONTROL_CONNECTION : int = 221
DATA_CONNECTION_OPEN_NO_TRANSFER_IN_PROGRESS : int = 225
CLOSING_DATA_CONNECTION_AFTER_SUCCESSFUL_ACTION : int = 226
ENTERING_PASSIVE_MODE : int = 227
USER_LOGGED_IN : int = 230
FILE_ACTION_OK_AND_COMPLETED : int = 250
DIRECTORY_CREATED : int = 257
USER_NAME_OK : int = 331
NEED_ACCOUNT_FOR_LOGIN : int = 332
REQUESTED_ACTION_PENDING_MORE_INFO : int = 350
SERVICE_NOT_AVAILABLE_CLOSING_CONTROL_CONNECTION : int = 421
CANNOT_OPEN_DATA_CONNECTION : int = 425
CONNECTION_CLOSED_TRANSFER_ABORTED : int = 426
ACTION_NOT_TAKEN_FILE_UNAVAILABLE_FILE_BUSY : int = 450
ACTION_ABORTED_LOCAL_ERROR : int = 451
ACTION_NOT_TAKEN_INSUFFICIENT_STORAGE_SPACE : int = 452
SYNTAX_ERROR_IN_COMMAND : int = 500
SYNTAX_ERROR_IN_PARAMETERS : int = 501
COMMAND_NOT_IMPLEMENTED : int = 502
BAD_SEQUENCE_OF_COMMANDS : int = 503
COMMAND_NOT_IMPLEMENTED_FOR_THAT_PARAMETER : int = 504
NOT_LOGGED_IN : int = 530
NEED_ACCOUNT : int = 532
ACTION_NOT_TAKEN_FILE_UNAVAILABLE : int = 550
ACTION_ABORTED_PAGE_TYPE_UNKNOWN : int = 551
ACTION_ABORTED_EXCEEDED_STORAGE_ALLOCATION : int = 552
ACTION_NOT_TAKEN_FILE_NAME_NOT_ALLOWED : int = 553
convertCode()

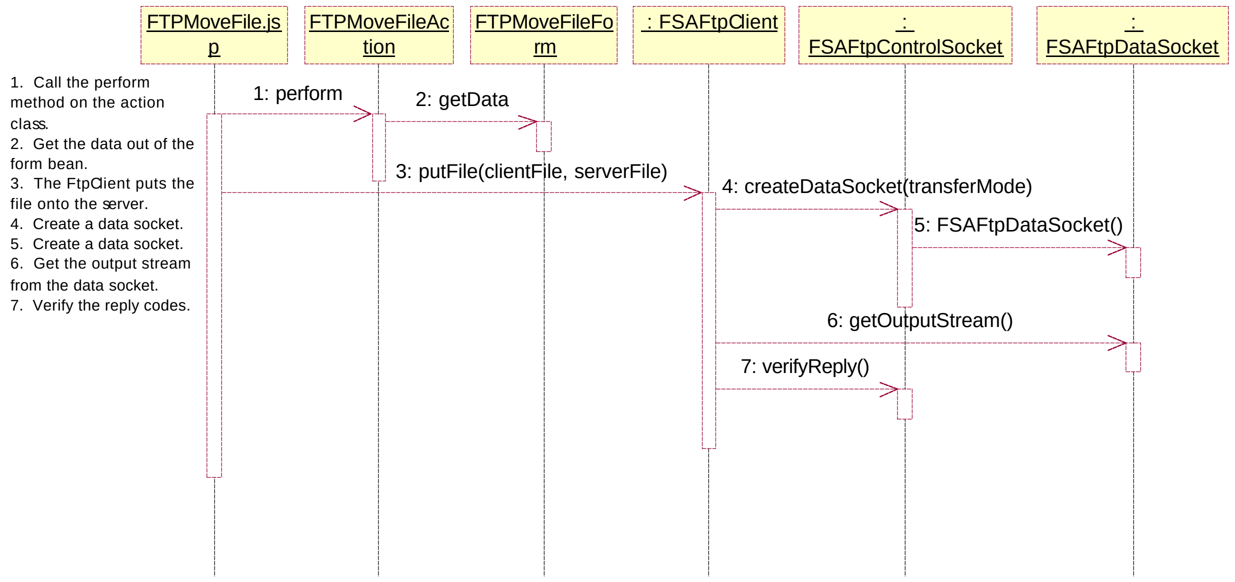
4.4.7 Sequence Diagrams

4.4.7.1.1 Make a FTP Connection


FTP Connection.jsp
FTPConnectAction
FTPConnectForm
: FSAFtpClient
: FSAFtpControlSocket
1: perform
2: getData
3: setTransferMode(transferMode)
4: connect(host, port, isSecure)
5: FSAFtpControlSocket(host, port, isSecure)

1. Call the perform method on the action class.
2. Get the data out of the form bean.
3. Set the transfer mode for the connection. This is active or passive.
4. The action class creates a FTP connection.
5. Create a control socket connection.

4.4.7.1.2 Get/Put a file



1. Call the perform method on the action class.
2. Get the data out of the form bean.
3. The FtpClient puts the file onto the server.
4. Create a data socket.
5. Create a data socket.
6. Get the output stream from the data socket.
7. Verify the reply codes.

4.4.8 References

- Sun Java Secure Socket Extension (JSSE) v. 1.0.2

  http://java.sun.com/products/jsse/index-102.html

- Overview, History, and Current Specification for FTP

  http://www.w3.org/Protocols/rfc959/

- Sun Java website

  http://java.sun.com

- Struts Framework website

  http://jakarta.apache.org/struts/

4.5 Configuration Framework

4.5.1 Purpose

The purpose of the configuration framework is to provide a standard for application configuration input and modification. The framework will allow configuration information to be loaded from properties files, XML files, or database tables.

The ITA configuration framework is implemented using the Accenture's General and Reusable Netcentric Delivery Solution (GRNDS) configuration framework. The GRNDS code has been extended to meet FSA application development requirements. Specifically, the framework has been extended to:

- Use a static initializer to load the configuration files, instead of using the GRNDS bootstrap framework.
- Support configuration input from database tables.

4.5.2 System Overview

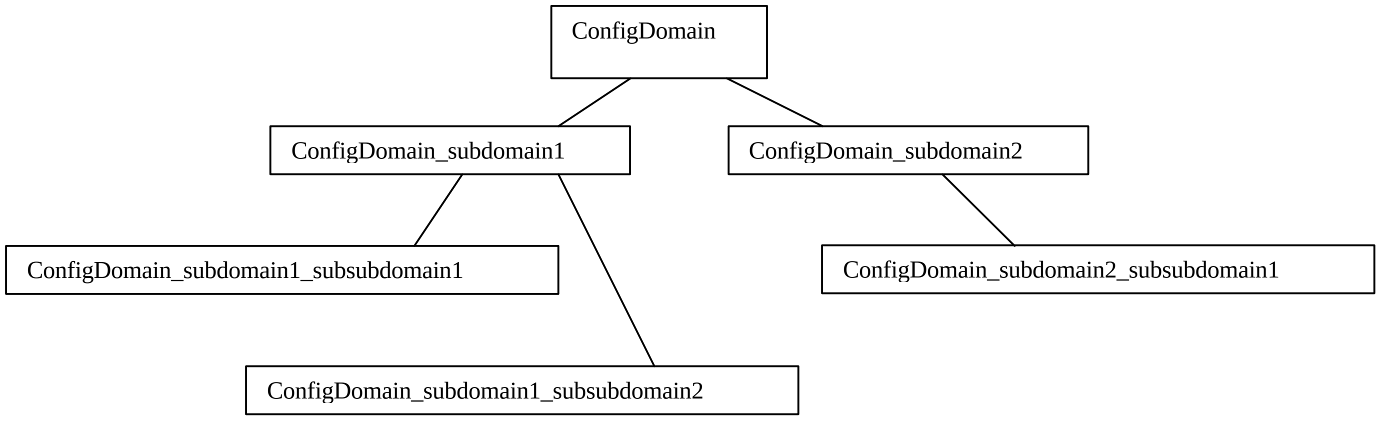
The configuration framework is created using the Java programming language. The configuration framework will run on IBM's WebSphere Application Server (WAS). The configuration framework provides a mechanism for the retrieval of configuration properties. The configuration framework allows configuration data to be stored in the form of properties files, XML files, database tables, or any combination of the three. The framework creates one common interface for application developers to access and manipulate this data.

The FSAConfiguration class contains a static initializer. The first time this class is loaded the static initializer is executed. The static initializer's responsibility is to load configuration data into memory. This is done using the Java ResourceBundle functionality.

The Java ResourceBundle functionality locates properties files within the Classpath. The ResourceBundle class is extended to create an FSAXmlResourceBundle class that locates XML files within the Classpath. These properties (or XML files) contain base configuration data.

The configuration framework organizes the configuration data into different domains. Domains determine the scope for a property. Domains enable the concept of information inheritance, in which common information is kept in one domain, and overridden for specialized domains. These specialized domains are often referred to as sub domains, while the parent domain is referred to as the master domain.

The following diagram depicts the relationship between the master domain (which must be named “ConfigDomain” for properties files and XML files) and the sub domains.


ConfigDomain
ConfigDomain_subdomain1
ConfigDomain_subdomain2
ConfigDomain_subdomain1_subsubdomain1
ConfigDomain_subdomain2_subsubdomain1
ConfigDomain_subdomain1_subsubdomain2

Once the master domains are known, the application developer simply needs to request the configuration property value from the configuration source class using the domain (or sub-domain) and the configuration property key. If a value is not found for a specific key in the requested domain, the configuration framework looks in its parent domain. This continues until the key is found or the master domain is reached. If the key is not found a null value is returned.

4.5.3 Design Considerations

4.5.3.1 Assumptions and Dependencies

It is assumed that the configuration framework will function in a J2EE application server environment. As the current production server for FSA is IBM’s WAS v. 3.5, the framework will be compiled using its required JDK version 1.2.2. It will also work with the current JavaServer Pages (1.1), Java Servlet (2.2), and Java Database Connectivity (2.0) specifications for this server. It will be built and tested on the Sun Solaris 2.6 and HP-UX 11.0 operating systems. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

Although an upgrade to and use of JDK v. 1.4 configuration classes was examined, it was concluded that it does not support multiple source configuration files. The framework provides a configuration interface and currently can handle properties files, XML files, and database tables.

Another consideration is the application server, WAS v. 3.5, that is currently in production. WAS v. 3.5 is using the 1.2.2 version of the JDK. Given that the configuration classes examined were from JDK v. 1.4, and that the WebSphere Application Server's is using JDK v. 1.2.2, it is to possible to utilize these classes until WAS is upgraded to the most release of JDK.

4.5.3.2 Goals and Guidelines

The goal of the configuration framework is to provide a simple and robust framework that may be applied by any FSA application team utilizing the Java development environment. The configuration framework abstracts the reading of configuration data into the architecture layer. This avoids duplicate logic written by different application developers. The configuration framework also abstracts the representation of the configuration data, which allows the format of the data to change without having to perform any code changes.

4.5.3.3 Development Methods

This framework will be developed using general object-oriented software development techniques. The standard class and sequence diagrams are provided in this document. These diagrams should assist developers who are unfamiliar with this framework.

4.5.4 System Architecture

4.5.4.1 Overview

The Configuration Framework provides the following services:

- Configuration Data Load: the configuration data files are loaded into static variables
- Configuration Data Retrieval: returns the value based on the domain name and the tag / key name

4.5.4.2 SubsystemArchitecture: FSAConfiguration

The FSAConfiguration class is the class that contains the static initializer. It will find and load the master domain data.

4.5.4.3 Subsystem Architecture: FSAXmlResourceBundle

The FSAXmlResourceBundle class extends the java.util.ResourceBundle class. This class locates XML files within the Classpath.

4.5.4.4 Subsystem Architecture: GrndsConfiguration

It is the concrete facility class that is used to retrieve configuration data from a variety of sources. This class's primary responsibility is to supply configuration data from a variety of sources in a uniform fashion.

4.5.4.5 Subsystem Architecture: GrndsConfigurationEnvironment

The GrndsConfigurationEnvironment class extends the java.util.Properties class. This class encapsulates two ideas in application configuration: properties and configuration classes. This class bridges both approaches to representing configuration information by extending the java.util.Properties class and offering the ability to get the configuration objects created from all sources.

4.5.4.6 Subsystem Architecture: GrndsConfigurationSource

The GrndsConfigurationSource interface defines the base interface that all concrete sources must implement. The GrndsSystemPropertySource class, GrndsPropertyFileSource class, GrndsXmlFileSource class, and GrndsDatabaseSource class extend this class.

4.5.4.7 Subsystem Architecture: GrndsSystemPropertySource

This class integrates System properties into the GRNDS configuration facility. The source returns the system properties for all domains and subdomains.

4.5.4.8 Subsystem Architecture: GrndsPropertyFileSource

GrndsPropertyFileSource defines a concrete configuration source based on Java property files.

4.5.4.9 Subsystem Architecture: GrndsXmlFileSource

GrndsXmlFileSource defines a concrete configuration source based on XML files.

4.5.4.10 Subsystem Architecture: GrndsDatabaseSource

GrndsDatabaseSource defines a concrete configuration source based on database tables. GrndsDatabaseSource utilizes the ITA Persistence Framework for database access.

4.5.4.11 Subsystem Architecture: GrndsConfigurationException

The GrndsConfigurationException may be thrown while processing the configuration environment. This exception will be wrapped by the RCS exception handling framework.

4.5.5 Detailed System Design

4.5.5.1 Component Definitions

4.5.5.1.1 FSAConfiguration

Class Name:	FSAConfiguration
Component:	Configuration
Description:	This class uses a static initializer to load the master domains for the XML files, properties files, and database tables.
Package:	gov.ed. fsa.ita.config
Superclass:	Object

Attribute	**Type**	**Description**
Public:		
m_xmlMasterDomain	static String	The master domain for the XML configuration source.
m_propertiesMasterDomain	static String	The master domain for the properties file configuration source.
m_databaseMasterDomain	static String	The master domain for the database table configuration source.

Con/Destructors	**Arguments** (Type, Name)	**Description**
FSAConfiguration	None	Default Constructor.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Private:			
locateXmlMasterDomain	None	String	This method will locate the XML master domain using the FSAXmlResourceBundle class.
locatePropertiesMasterDomain	None	String	This method will locate the properties file master domain using the PropertiesResourceBundle class.
locateDatabaseMasterDomain	None	String	This method will locate the database master domain.

4.5.5.1.2 FSAXmlResourceBundle

Class Name:	FSAXmlResourceBundle
Component:	Configuration
Description:	This class locates XML files within the Classpath.
Package:	gov.ed. fsa.ita.config
Superclass:	java.util.ResourceBundle

Con/Destructors	Arguments (Type, Name)	Description
FSAXmlResourceBundle	none	Default Constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
getKeys	none	java.util.Enumeration	Implementation of ResourceBundle.getKeys().
handleGetObject	String key	Object	Get an object from a ResourceBundle.

4.5.5.1.3 GrndsConfiguration Interface

Interface Name:	GrndsConfiguration
Component:	Configuration
Description:	This interface defines what methods are required for a functional configuration source.
Package:	org.grnds.facility.config
Superclass:	org.grnds.foundation.GrndsObject

Attribute	Type	Description
Public:		
ms_instance	static GrndsConfiguration	The instance of this configuration.
GRNDS_CONFIG_DOMAIN	static final String	Default configuration domain.
m_isInitialized	boolean	True if the configuration is initialized. False if the configuration is not initialized.
m_sources	java.util.Vector	The sources loaded in this configuration.
m_envCache	org.grnds.foundation.cache..GrndsCache	The cache storing the configurations. (The ITA configuration framework will not be using this functionality).

Con/Destructors	Arguments (Type, Name)	Description
GrndsConfiguration	none	Default Constructor.
GrndsConfiguration	GrndsConfigurationSource[] srcs_	Constructor that creates a logging facility with the set configuration sources.
GrndsConfiguration	GrndsConfiguration rhs_	Copy Constructor.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Private:			
doInitSources	final String[] args_	void	This method loads and initializes log categories into the logging facility.
doFiniSources	none	void	This method finalizes and removes log categories from the logging facility.
makeExpirationPlan	none	GrndsCacheExpirationPlan	
Protected:			
createKey	String domain_, String[] subdomains_	static final String	Returns the unique key derived from the given configuration domain and subdomains.
findCachedEnvironment	String domain_, String[] subdomains_	final synchronized GrndsConfigurationEnvironm ent	Returns the cached environment for the given configuration domain and subdomains. (The ITA configuration framework will not be using this functionality).
doToString	none	String	Stream the tags registered with the trace facility.
Public:			
getInstance	none	static final synchronized GrndsConfiguration	Returns the singleton instance for the configuration facility.
parseSubdomainArray	String subdomainList_	String []	This method parses the subdomain list and returns a String array containing all the subdomains.
setInstance	GrndsConfiguration newInstance_	void	Sets the singleton instance for the GrndsConfiguration facility.
clone	None	Object	Clones the current GrndsConfiguration
contains	String domain_, String key_	final boolean	Returns true if there exists at least one configuration source that contains the property identified by key_.
contains	String domain_, String[] subdomains_, String key_	boolean	Returns true if there exists at least one configuration source that contains the property identified by key_.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
getEnvironment	String domain_	GrndsConfigurationEnvironment	Returns a Properties set reflecting the entire configuration environment.
getProperty	String domain_, String[] subdomains_, String key_	String	Returns the property within the domain_/subdomains_ corresponding to the given key_.
init	String[] args	synchonized void	Initializes the Logging Facility prior to use. First, the Configuration facility is used to load the log category / channel baseline. Each loaded category and channel are then initialized.
fini	None	synchonized void	Perform finalization operations for the Logging facility. This operation results in the finalization of all the logging categories and the closing of their corresponding PrintStreams.
refresh	None	synchonized void	Refreshes the entire configuration environment.
addSource	org.grnds.facility.config.GrndsConfigurationSource src_	void	Adds a new log category / pair to the logging facility. If the category was already registered with the logging facility, then the previous association pair is replaced with the given pair.
getSources	None	Enumeration	Returns an enumeration that provides access to all of the configuration sources.

4.5.5.1.4 GrndsConfigurationEnvironment Class

Class Name:	GrndsConfigurationEnvironment
Component:	Configuration
Description:	This class encapsulates two ideas in application configuration: properties and configuration classes.
Package:	org.grnds.facility.config

Superclass:	java.util.Properties

Attribute	**Type**	**Description**
Public:		
m_objects	java.util.Hashtable	The hashtable of key and value pairs.

Con/Destructors	**Arguments** (Type, Name)	**Description**
GrndsConfigurationEnvironment	none	Default Constructor.
GrndsConfigurationEnvironment	GrndsConfigurationEnvironment rhs_	Copy Constructor.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
clone	none	Object	Clones the current GrndsConfigurationEnvironment.
hashCode	none	int	Returns the hashcode of the m_object attribute.
equals	Object rhs_	boolean	Returns true if the 2 GrndsConfigurationEnvironments are logically equal.
putAll	Properties from_	void	Put the keys in the hashtable
getConfigurationObject	Class type_	Object	Returns the configuration object based on Class type_.
getConfigurationObject	String type_	Object	Returns the configuration object based on String type_.
getConfigurationObjects	none	Enumeration	Return all configuration objects
addConfigurationObject	Object config_	void	Adds a configuration object to the hashtable
toString	none	String	This method is called from the template toString method. Subclasses implement doToString() to help produce an appropriate string representation of the object.
getSources	none	Enumeration	Returns an enumeration that provides access to all of the configuration sources.

4.5.5.1.5 GrndsConfigurationSource Interface

Interface Name:	GrndsConfigurationSource
Component:	Configuration
Description:	This interface defines the base interface that all concrete sources must implement.
Package:	org.grnds.facility.config
Superclass:	java.lang.object

Attribute	Type	Description

Con/Destructors	Arguments (Type, Name)	Description

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
clone	None	abstract Object	Clones the current GrndsConfigurationSource.
getEnvironment	String domain_, String[] subdomains_	abstract GrndsConfigurationEnvironment	Returns a Properties set reflecting the configuration environment defined by this source.
init	String[] args_	abstract void	Concrete facility classes define this method to perform initialization operations. The input arguments is the args_ array provided to main().
fini	None	abstract void	Concrete application classes define this method to perform finalization operations.
refresh	None	abstract void	Refreshes the configuration source. This method enables the ability to dynamically update configuration information without restarting the JVM.

4.5.5.1.6 GrndsSystemPropertySource Class

Class Name:	GrndsSystemPropertySource
Component:	Configuration
Description:	This class integrates System Properties into the grnds configuration facility. It implements the GrndsConfigurationSource interface.
Package:	org.grnds.facility.config
Superclass:	org.grnds.foundation.GrndsObject

Attribute	Type	Description

Con/Destructors	Arguments (Type, Name)	Description
GrndsSystemPropertySource	none	Default Constructor
GrndsSystemPropertySource	GrndsSystemPropertySource rhs_	Copy Constructor

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
clone	none	abstract Object	Clones the current GrndsSystemPropertySource
getEnvironment	String domain_, String[] subdomains_	GrndsConfigurationEnvironment	Returns a Properties set reflecting the configuration environment defined by this source.
init	String[] args_	abstract void	Concrete facility classes define this method to perform initialization operations. The input arguments is the args_ array provided to main().
fini	none	abstract void	Concrete application classes define this method to perform finalization operations.
getPropertyNames	String domain_, String[] subdomains_	Enumeration	Returns an enumeration to all of the available property names contained in this configuration source.
refresh	none	abstract void	Refreshes the configuration source. This method enables the ability to dynamically update configuration information without restarting the JVM.

4.5.5.1.7 GrndsPropertyFileSource Class

Class Name:	GrndsPropertyFileSource
Component:	Configuration
Description:	This class defines a concrete configuration source based on Java property files. It implements the GrndsConfigurationSource interface.
Package:	org.grnds.facility.config
Superclass:	org.grnds.foundation.GrndsObject

Attribute	**Type**	**Description**

Con/Destructors	**Arguments** (Type, Name)	**Description**
GrndsPropertyFileSource	ClassLoader loader_	Primary Constructor - builds a GrndsPropertyFileSource instance with the given ClassLoader object.
GrndsPropertyFileSource	GrndsPropertyFileSource rhs_	Copy Constructor

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Private:			
createSubDomainString	String[] subdomains_	static String	Returns a StringBuffer populated with a leading DOMAIN_SEPARATOR, followed by each subdomain separated by DOMAIN_SEPARATOR. An empty StringBuffer object is returned if either subdomains_ is null or contains zero elements.
findEnvironment	String domain_, String subdomains_, ClassLoader loader_	Properties	Finds the environment properties file.
getResourceAsStream	String name_, ClassLoader loader_	InputStream	Returns the resource as an input stream.
Public:			
clone	none	abstract Object	Clones the current GrndsSystemPropertySource.
getEnvironment	String domain_, String[] subdomains_	GrndsConfigurationEnvironment	Returns a Properties set reflecting the configuration environment defined by this source.
init	String[] args_	abstract void	Concrete facility classes define this method to perform initialization operations. The input arguments is the args_ array provided to main().
fini	none	abstract void	Concrete application classes define this method to perform finalization operations.
getPropertyNames	String domain_, String[] subdomains_	Enumeration	Returns an enumeration to all of the available property names contained in this configuration source.
refresh	none	abstract void	Refreshes the configuration source. This method enables the ability to dynamically update configuration information without restarting the JVM.

4.5.5.1.8 GrndsXmlFileSource Class

Class Name:	GrndsXmlFileSource
Component:	Configuration
Description:	This class defines a concrete configuration source based on XML files.
Package:	org.grnds.facility.config
Superclass:	org.grnds.foundation.GrndsObject

Attribute	Type	Description
Private:		
m_loader	ClassLoader	The class loader for the XML file.
m_configDocument	String	The configuration document for the XML file.
m_configMap	Hashtable	dialect, SourceEntry
m_saxDriverClass	String	The SAX driver to be used to parse the XML
m_validateDocuments	boolean	True is the document should be validated. False if the document should not be validated.
m_isInitialized	boolean	True if the class has been initialized. False if the class has not been initialized.

Con/Destructors	Arguments (Type, Name)	Description
GrndsXmlFileSource	String configMap_, ClassLoader loader_	Builds a GrndsXmlFileSource instance with the config map identifying the XML configuration map that associates XML configuration documents with configuration domains/subdomains. The default GrndsXmlReader SAX driver is used and document validation is turned off.
GrndsXmlFileSource	String configMap_, String saxDriverClass_, ClassLoader loader_	Builds a GrndsXmlFileSource instance with the config map. The second argument identifies the desired XML SAX parser. Document validation is turned off.
GrndsXmlFileSource	String configMap_, boolean validateDocuments_, ClassLoader loader_	Builds a GrndsXmlFileSource instance with the config map. The default GrndsXmlReader SAX driver is used. The second argument directs whether validation should be performed.
GrndsXmlFileSource	String configMap_, String saxDriverClass_, boolean validateDocuments_, ClassLoader loader_	Builds a GrndsXmlFileSource instance with the config map. The second and third arguments identify the desired XML SAX parser and whether validation should be performed.
GrndsXmlFileSource	GrndsXmlFileSource rhs_	Copy Constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
initConfigurationMap	none	void	Initializes the configuration map
setSourceEntry	String key_, SourceEntry newEntry_	void	Set the entry source for the configuration map.
parseDomainEnvironment	Element domainEnv_, String basename_	void	Parses the domain values in the XML file.

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
populateDomainEntries	GrndsConfigurationEnvironment env_, Enumeration domainEntries_	void	Populate the domain entries into the environment.
getSourceLoadStrategy	Element source_	SourceLoadStrategy	Retrieve the source load strategy for the XML file.
Public:			
clone	none	abstract Object	Clones the current GrndsXmlFileSource object
setFileSourceDialect	String domain_, String[] subdomains_, String absoluteFilename_, Class dialect_	void	Sets the XML dialect and file source (by absolute filename) for the given configuration domain/subdomains.
setResourceSourceDialect	String domain_, String[] subdomains_, String resourceName_, Class dialect_, ClassLoader cl_	void	Sets the XML dialect and source (by class loader resource) for the given configuration domain/subdomains.
setUrlSourceDialect	String domain_, String[] subdomains_, String resourceUrl_, Class dialect_	void	Sets the XML dialect and URL source for the given configuration domain/subdomains.
getEnvironment	String domain_, String[] subdomains_	GrndsConfiguration Environment	Returns a configuration environment object reflecting the configuration environment defined by this source.
init	String[] args_	void	Initializes the XML file source
fini	none	void	Finalizes the XML file source.
refresh	none	synchronized void	Refreshes the configuration source.
doToString	none	String	Coverts object to string.
buildSourceKey	String domain_, String[] subdomains_	static String	Builds the source key for the XML file.

4.5.5.1.9 GrndsDatabaseSource Class

Class Name:	GrndsDatabaseSource
Component:	Configuration
Description:	This class defines a concrete configuration source based on database tables. It implements the GrndsConfigurationSource interface.
Package:	org.grnds.facility.config
Superclass:	GrndsObject

Con/Destructors	Arguments (Type, Name)	Description
GrndsDatabaseSource	ClassLoader loader_	Primary Constructor - builds a GrndsDatabaseSource instance with the given ClassLoader object.
GrndsDatabaseSource	GrndsDatabaseSource rhs_	Copy Constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Private:			
createSubDomainString	String[] subdomains_	static String	Returns a StringBuffer populated with a leading DOMAIN_SEPARATOR, followed by each subdomain separated by DOMAIN_SEPARATOR. An empty StringBuffer object is returned if either subdomains_ is null or contains zero elements.
findEnvironment	String domain_, String subdomains_, ClassLoader loader_	Properties	Finds the environment properties file.
Public:			
clone	none	abstract Object	Clones the current GrndsDatabaseSource.
getEnvironment	String domain_, String[] subdomains_	GrndsConfigurationEnvironment	Returns a Properties set reflecting the configuration environment defined by this source.
init	String[] args_	abstract void	Concrete facility classes define this method to perform initialization operations. The input arguments is the args_ array provided to main().
fini	none	abstract void	Concrete application classes define this method to perform finalization operations.
getPropertyNames	String domain_, String[] subdomains_	Enumeration	Returns an enumeration to all of the available property names contained in this configuration source.
refresh	none	abstract void	Refreshes the configuration source. This method enables the ability to dynamically update configuration information without restarting the JVM.

4.5.5.1.10 GrndsConfigurationException

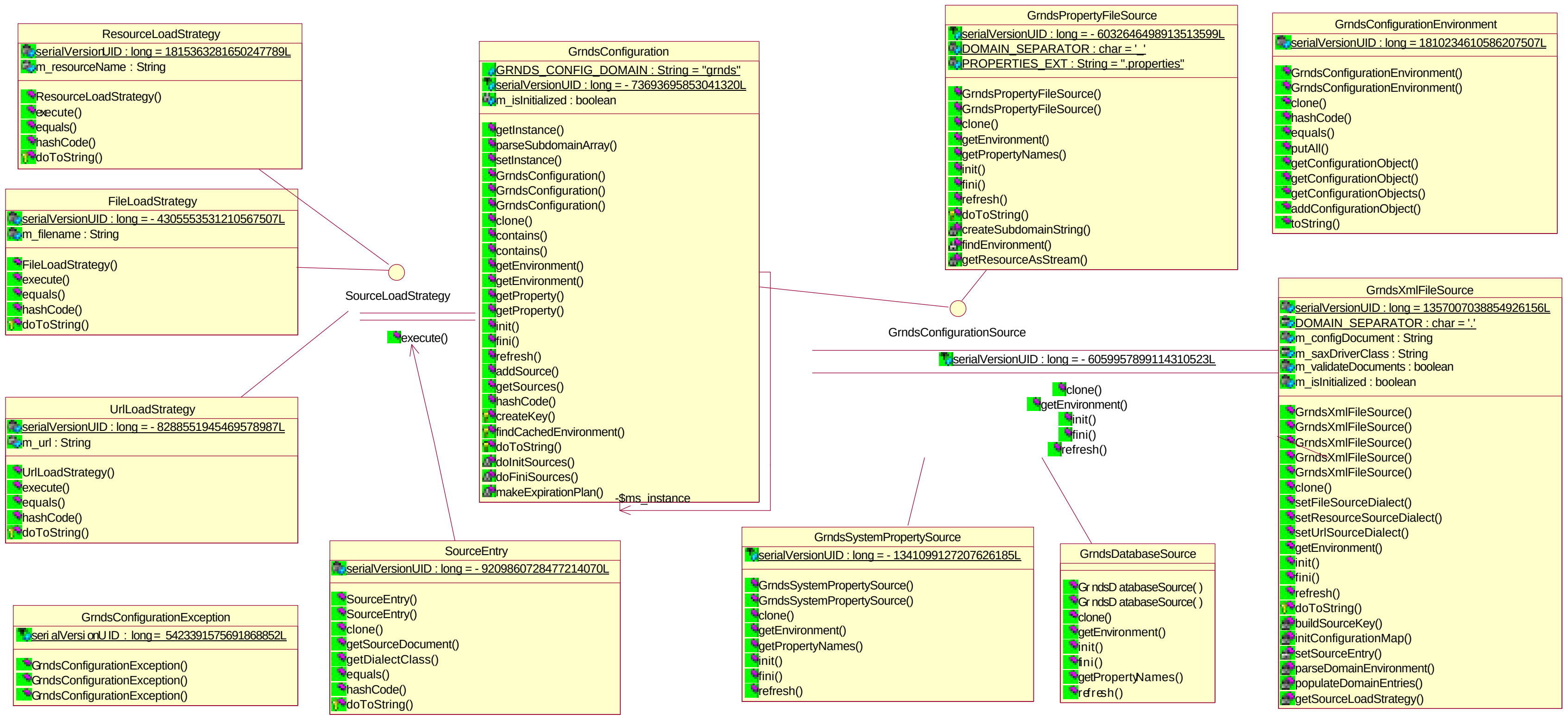
Class Name:	GrndsConfigurationException
Component:	Configuration
Description:	This class may be thrown while processing the configuration environment.
Package:	org.grnds.facility.config
Superclass:	org.grnds.foundation.exception.GrndsRuntimeException

Attribute	**Type**	**Description**

Con/Destructors	**Arguments** (Type, Name)	**Description**
GrndsConfigurationException	none	Default Constructor.
GrndsConfigurationException	String msg_	Creates GrndsConfigurationException with a message.
GrndsConfigurationException	String msg_, Throwable error_	Creates GrndsConfigurationException with a message an root exception.

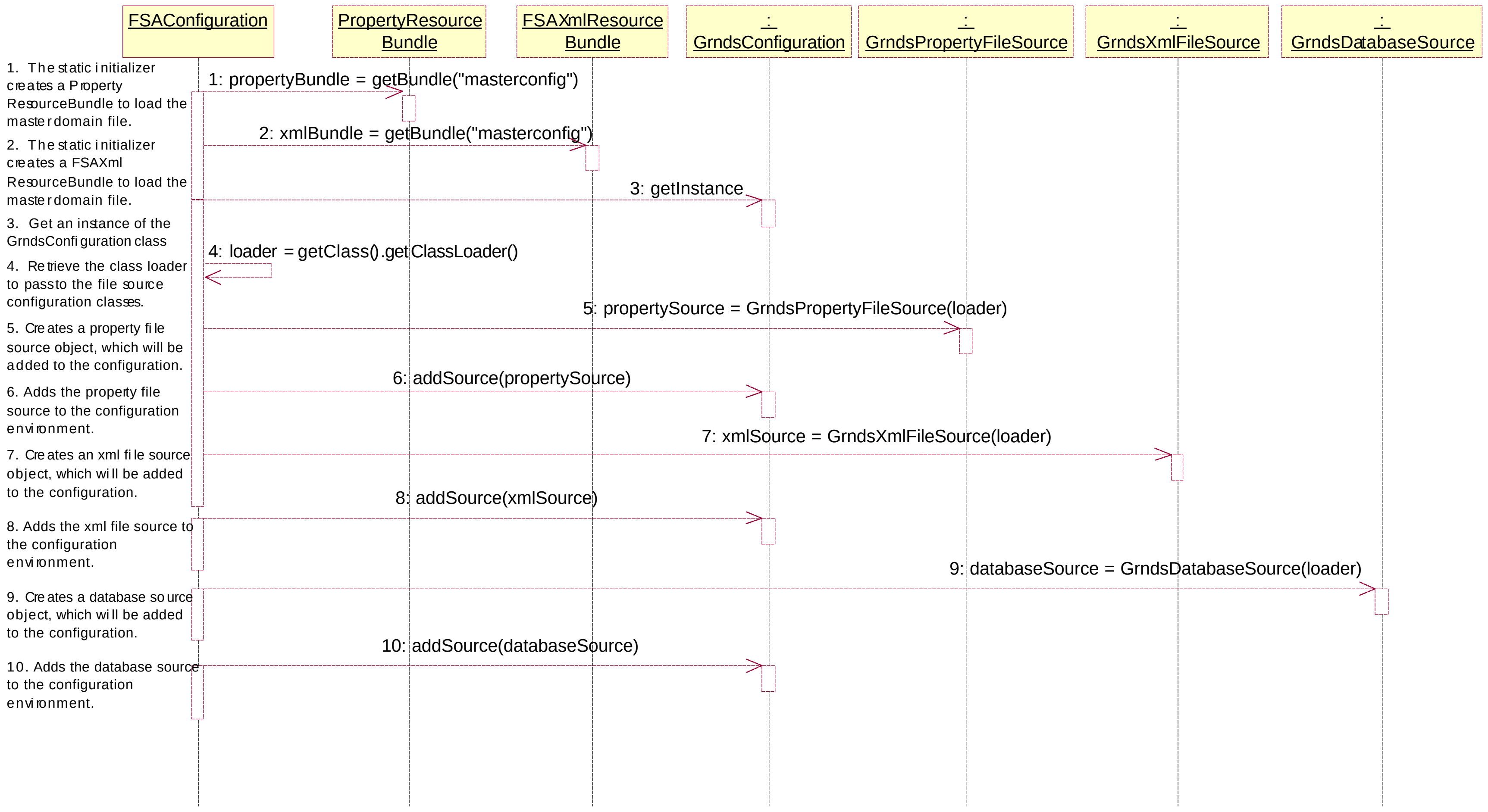
4.5.6 Class Diagram

4.5.6.1 Configuration Classes



4.5.7 Sequence Diagram

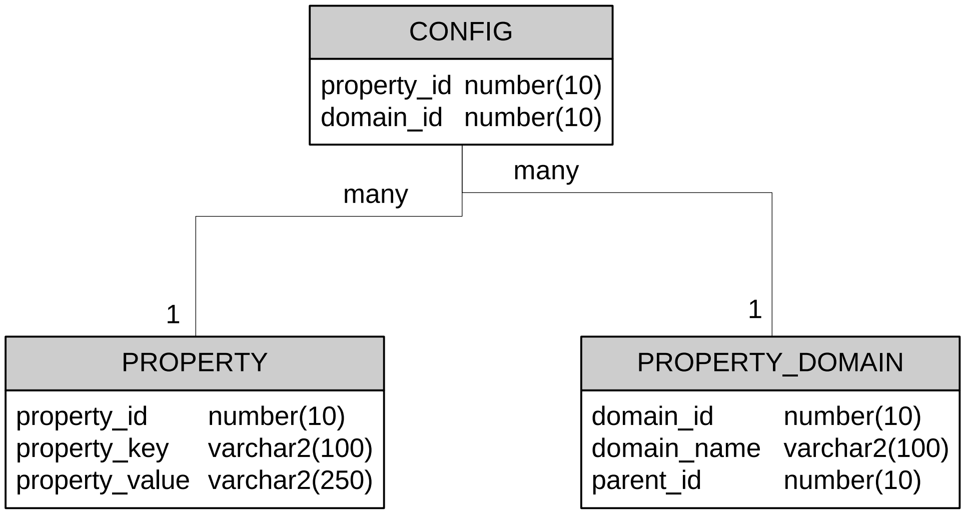
4.5.7.1.1 Static Initializer


FSAConfiguration
PropertyResource Bundle
FSAXmlResource Bundle
: GrndsConfiguration
: GrndsPropertyFileSource
: GrndsXmlFileSource
: GrndsDatabaseSource
1: propertyBundle = getBundle("masterconfig")
2: xmlBundle = getBundle("masterconfig")
3: getInstance
4: loader = getClass().getClassLoader()
5: propertySource = GrndsPropertyFileSource(loader)
6: addSource(propertySource)
7: xmlSource = GrndsXmlFileSource(loader)
8: addSource(xmlSource)
9: databaseSource = GrndsDatabaseSource(loader)
10: addSource(databaseSource)

1. The static initializer creates a Property ResourceBundle to load the masterdomain file.
2. The static initializer creates a FSAXml ResourceBundle to load the masterdomain file.
3. Get an instance of the GrndsConfiguration class
4. Retrieve the class loader to pass to the file source configuration classes.
5. Creates a property file source object, which will be added to the configuration.
6. Adds the property file source to the configuration environment.
7. Creates an xml file source object, which will be added to the configuration.
8. Adds the xml file source to the configuration environment.
9. Creates a database source object, which will be added to the configuration.
10. Adds the database source to the configuration environment.

4.5.8 Data Model

This diagram depicts the data model that will be used for configuration parameters stored in database tables. The domain id of 1 will be considered the master domain.



4.5.9 References

- GRNDS Framework

  https://onesource.accenture.com

- Sun Java website

  http://java.sun.com

4.6 XML Helper

4.6.1 Purpose

This section covers the components that comprise the XMLHelper Framework. The XMLHelper framework includes custom code by the RCS developers as well as code provided by the Accenture's GRNDS initiative and the open-source framework Castor -- (http://www.castor.org).

GRNDS is an Accenture led initiative to build reusable Java based frameworks that can be customized to fit various requirements. In this case GRNDS provides the ability to read XML documents using the Document Object Model (DOM) API. The GRNDS provides propertiesPlus Object, uses the DOM API authored by the W3C standards group to read and manipulate XML documents. The DOM API standard aims to make a platform and language neutral program interface to documents so that programmers may focus on their programming model and not worry about document reading or editing.

Castor provides immediate instantiation of Java objects directly from XML documents and allows developers to build XML documents from in-memory Java objects. Castor XML is an XML data-binding framework that is unlike the two main XML APIs, DOM (Document Object Model) and SAX (Simple API for XML). The DOM and SAX API's deal with the structure of an XML document, Castor enables manipulation of the data defined in an XML document through an object model which represents that data.

While both of these frameworks make use of many Java features and packages that are provided by the Sun Java Development kit 1.2.2, these topics are not covered in depth. Consult the Sun web site for more detailed information.

4.6.2 System Overview

In the past, FSA applications have used XML documents in various ways. FSA applications have had a need to parse XML documents for application initialization, error messages, and argument or parameter changes. In solving these issues, several standard XML parsers have become known in the last few years. The Document Object Model (DOM) and the Simple API for XML are currently the two most popular API's for manipulating XML documents. The DOM API, which was developed by the standards group, the World Wide Web Consortium (W3C), builds an in-memory tree like structure that represents the XML document. The DOM API allows developers to move up and down the branches of the DOM tree requesting and reviewing data.

The SAX API uses an event-based stream to parse the XML document. As the SAX API uncovers XML tokens, it presents these tokens to the parser. The parser then generates events at known points within the document. It is up to the developer to decide what to do when these events are generated. An example of a SAX event would be at the start and end of a document, the start and end of an element when it finds characters within the element. Both API's have their strengths and weaknesses.

In the development process, the DOM parser would be implemented when

- The XML document's structure must be known
- Parts of the XML document must be manipulated or moved within the document, such as a bubble sort
- The XML document's information needs to used more than once

In the development process, the SAX parser would be implemented when

- Extreme machine limitations such as memory or CPU exist
- A few elements of the XML need to be extracted
- Using the XML document's information only once

The SAX Parser is pressed into duty in instances where a search for key words within an XML document is necessary. For example, if a sort for all elements within an XML document were to be conducted, then holding the DOM tree within memory is a reasonable technique; however, if the desired result were to find the number of occasions in which a word appeared in an XML document, then holding the entire DOM tree within memory is not a sensible measure.

The main purpose of the DOM and SAX API's is to access and manipulate the structure of an XML document. Another need within the XML community is to manipulate the data defined within the XML document. Several products have emerged that perform data binding from an XML document to a Java Object. Castor and Java Architecture for XML Binding (JAXB) are both Java API's that are leading the charge for XML Data Binding frameworks. These frameworks will compile and build Java objects from XML documents, as well as create of a Java object in memory and output the equivalent XML document. JAXB is Sun's interpretation for Data binding. This specification is very new and has not been finalized. The final draft of JAXB v1.0 is due near the end of 2002. Castor is the parent of JAXB and as such is much more stable and encompasses a larger number of features.

4.6.3 Design Considerations

4.6.3.1 Assumptions and Dependencies

It is assumed that this framework will function in a J2EE application server environment. As the current production server for FSA is IBM WebSphere 3.5.X, the framework will be compiled using its required JDK version 1.2.2. It should also work with the current JavaServer Pages (1.1), Java Servlet (2.2), and Java Database Connectivity (2.0) specifications for this server. It will be built and tested on Sun Solaris 2.6 and HP-UX 11.0. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.6.3.2 Goals and Guidelines

The goal of this development is to provide a simple yet robust XMLHelper Framework that encompasses the three different parsing faculties specified above. Any FSA development team building applications in a Java environment can easily utilize these parsing capabilities. Most FSA applications teams are implementing some type of XML parser that helps simplify the development and maintenance of respective applications. By establishing the standard of a single XML helper framework, development cycles are abbreviated and best practices for XML data manipulation are included in the framework to be used by all development teams.

The FSA Application Operations Groups needs to have the ability to trace and debug problems related to the XML Helper Framework. To this end the ITA RCS Logging and Exception Handling Framework will be used to document any errors that the XMLHelper Framework encounters.

A final goal is to ensure that the performance of the XMLHelper Framework does not hinder the performance of the application using it. Part of the release process of ITA RCS Frameworks is to ensure optimal performance. This is accomplished by conducting performance tests and documenting performance results.

4.6.3.3 Development Methods

The development of this framework entails the use of general object-oriented software development techniques as specified in any standard text on the Java programming language. The design of this framework's class and relationship patterns did not require the use of an object oriented modeling tool or methodology. However, the resulting class files have been documented with standard class diagrams and sequence diagrams using Rational Rose in order to illustrate their structure more readily. These diagrams (included below) will assist programmers unfamiliar with this framework.

4.6.4 System Architecture

The RCS XML Helper Framework makes use of a wrapper class called FSAXMLHelper that will encompass several XML parsing classes that implement DOM Level 2 parsing, SAX 2.0 parsing and XML data-binding.

4.6.4.1 Subsystem Architecture: FSAXMLHelper

This class provides a simplified interface to the entire XML Helper framework. Several parse methods are provided depending on whether the developer requires the use of DOM parsing, SAX parsing or Castor Java Binding. This class inherits from the FSADomXml class to gain access to the many methods already deployed within FSADomXml. The following code sample illustrates the different parse methods and what architecture they are tied to.

```
FSAXMLHelper xmldom = FSAXMLHelper();
FSAXMLHelper xmlsax = FSAXMLHelper();
FSAXMLHelper xmlbind = FSAXMLHelper();
// illustration of a DOM parse
xmldom.parse("/www/test/properties/dom.xml") ;
// illustration of a SAX parse
xmlsax.parse("/www/test/properties/sax.xml",handler);
//illustration of a DataBind parse
xmlbind.parse("/www/test/properties/mapping.xml"," /www/test/properties/data.xml");
```

4.6.4.2 Subsystem Architecture: FSADomXml

This is the FSA class that implements DOM levels 1 and DOM levels 2 of the W3C specifications for parsing XML. The W3C specifications are final and can be implemented without concern for future specification changes. While the DOM standard defines many XML components, these are the most commonly used:

- Element: Elements consist of a start tag, an end tag and the content in between. The content is text based inside the XML document but once parsed it can be converted to anything

- Attribute: Elements use attributes to help describe the element. The attribute needs to be quoted. <car color="green">volvo</car>

- Document: Represents the entire XML document

Besides parsing the XML document, the ITA XMLHelper framework simplifies the retrieval of internal elements and the element text values by storing them within a hash table that is within a FSADomXml object. If an element contains other elements within it, then the deeper elements will be within another FSADomXml object that is embedded within the original FSADomXml object.

```
<?xml version="1.0"?>
<Arguments Number="1">
<Arg0 Type="String" Content="Hello There"> </Arg0>
<Arg1 Type="integer" Content="1"> </Arg1>
<Arg2 Type="List" Content="one two three four"> </Arg2>
<Arg3>Hello</Arg3>
</Arguments>
```

In the previous XML file, Arg0, Arg1, Arg2 and Arg3 are recognized as element keys and are stored in the FSADomXml hash table. Number is an attribute key but is still within the same FSADomXml hash table. If a developer needed access to the attribute key Type, which is below the key Arg0, then that developer would have to drill down on the PropertiesPlus object using the FSADomXml public method *getNextDomObject(key)* and passing the key Arg0. This will pass back the FSADomXml object that has the hash table that holds the value for Type and Content. Then using the public methods *getString*, the developer could get the value for Type.

4.6.4.3 Subsystem Architecture: FSASaxXml

This is the FSA class that implements the SAX API. The SAX API initiates an event handler when an event occurs. Common Events are:

- StartDocument
- EndDocument
- StartElement
- EndElement
- Character

When these events occur, the SAX API launches the respective handler method that corresponds to the event. A developer will use the FSASaxApi by building a customized handler class and then inputting that handler into the public *parse(xml,handler)* method. This allows the developer to customize the FSA Sax API to fit the application needs. An example of a customized handler is below. The customized handler class is called FSASaxHandlers and it must be a subclass of FSASaxHandler.

```
FSAXMLHandler it = new FSAXMLHandler();
        try {
                FSASaxHandlers handler = new FSASaxHandlers();
        it.parse("d:/www/test/properties/example.xml",handler);
        } catch (Exception e) {
                System.err.println(e);
                }
        }

package gov.ed.fsa.ita.schedule.example;
import gov.ed.fsa.ita.xmlhelper.*;
import gov.ed.fsa.ita.schedule.*;
import gov.ed.fsa.ita.schedule.example.*;
import java.util.*;
import java.io.*;
import java.lang.*;
public class FSASaxHandlers extends FSASaxHandlers{
public FSASaxHandlers() {
        super();
}

public void endDocument() {
        System.out.println("End document in customized handler class");}

public void startDocument() {
        System.out.println("Start document in customized class");
        }
}
```

4.6.4.4 Subsystem Architecture: FSADataBind

The FSADataBind class provides a data-binding ability, which allows developers to deal with data that is defined within XML document through an object model, which represents the data. The FSADataBind class can marshal any bean-like Object from and to XML. In all cases the framework uses class and field descriptors that are described within a separate mapping XML document that defines the attributes of the class. This mapping XML document defines the type of the attribute and how the attribute should be loaded. The attributes can be populated either by a value directly inputted from the XML document or may be populated by a get method. A second XML document actually defines the values of the Java Object. Thus by calling the public method parse (*mapping.xml,data.xml*), a developer can instantiate a Java Object defined within the data XML file, as demonstrated by the following example.

```
FSAXMLHelper xmlbind = new FSAXMLHelper();
        FSAScheduleEntry E = new FSAScheduleEntry();
  E = (FSAScheduleEntry) xmlbind.parse("/www/test/properties/mapping.xml",
"/www/test/properties/schedule.xml");
```

In this example, the class and attribute definitions are defined within the mapping.xml and the actual values are defined within schedule.xml. The actual XML documents are below.

```
<?xml version="1.0"?>
<!DOCTYPE mapping PUBLIC "-//EXOLAB/Castor Object Mapping DTD Version 1.0//EN" "mapping.dtd">
<mapping>
        <class name="gov.ed.fsa.ita.schedule.FSAScheduleEntry">
                <map-to xml="SEntry"/>
                <field name="m_classname"
                        type="java.lang.String"
                        direct="true">
                        <bind-xml name="Classname" node="element"/>
                </field>
                <field name="m_methodname"
                        type="java.lang.String"
                        direct="true">
```

```
                              <bind-xml name="Methodname" node="element"/>
                        </field>
            </class>
</mapping>

<SEntry>
                        <Classname>gov.ed.fsa.ita.schedule.test.SchFire</Classname>
                        <Methodname>method2</Methodname>
</SEntry>
```

4.6.4.5 SubSystem Architecture: ITA RCS Logging and Exception Handling Framework

The RCS Logging and Exception Handling frameworks have been added to the RCS XMLHelper framework to enhance debugging and tracing abilities. This should benefit operations ability to isolate problems that may occur within the framework.

4.6.5 Detailed System Design

4.6.5.1 Component Definitions

4.6.5.1.1 FSAXMLHelper

Class Name:	FSAXMLHelper
Component:	XML Helper
Description:	Public class
Package:	gov.ed. fsa.ita.xmlhelper
Superclass:	FSADomXml

Con/Destructors	**Arguments** (Type, Name)	**Description**
FSAXMLHelper	none	Constructor

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
getNextDomObject	(Object key)	FSADomXml throws FSAException	Looks for next FSADomXML object that would exist inside present one.
parse	(String mapping, String data)	Void throws FSAException	Instantiated XML file into Java Objects using a mapping file to describe the object and the data file to describe the values of the java object.
parse	(String saxfile, SchSaxHandlers handler)	Void throws FSAException	Uses the SAX API parser to parse an XML Document using level 2 Sax.
parse	(String domfile)	Void throws FSAException	Uses the DOM API parser to parse an XML Document using level 2 Dom.

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
write	(Object classname, String maploc, String writeloc)	Void throws FSAException	Using the castor framework, will marshal a Java object to a XML document.

4.6.5.1.2 FSADomXml

Class Name:	FSADomXml
Component:	XML Helper
Description:	Public class
Package:	gov.ed. fsa.ita.xmlhelper
Superclass:	java.util.HashTable

Con/Destructors	Arguments (Type, Name)	Description
FSADomXml	none	Constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
attributeKeys	None	Enumeration Throws FSAException	Returns a enumeration of Attribute Keys relating to current FSADomXml Object.
elementKeys	None	Enumeration Throws FSAException	Returns a enumeration of Element Keys relating to current FSADomXml Object.
GetList	(String key)	String Throws FSAException	Returns the value of the key element in a list format.
GetString	(String key)	Void Throws FSAException	
GetObject	(String key)	Void Throws FSAException	

4.6.5.1.3 FSASaxXml

Class Name:	FSASaxXml
Component:	XML Helper
Description:	Public class
Package:	gov.ed. fsa.ita.xmlhelper
Superclass:	DefaultHandler

Con/Destructors	Arguments (Type, Name)	Description
FSASaxXml	none	Constructor

Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
Parse	(String xmlfile)	Void Throws FSAException	Parseas XML document using the SAX API and default handlers.
Parse	(String xmlfile,FSASaxHandlers)	Void Throws FSAException	Parses XML document using the SAX API and user custom handlers.

4.6.5.1.4 FSADataBind

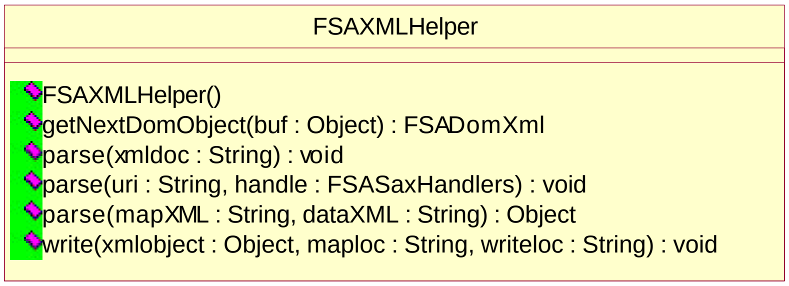
Class Name:	FSADataBind
Component:	XML Helper
Description:	Public class
Package:	gov.ed. fsa.ita.xmlhelper
Superclass:	Object

Con/Destructors	Arguments (Type, Name)	Description
FSADataBind	none	Constructor

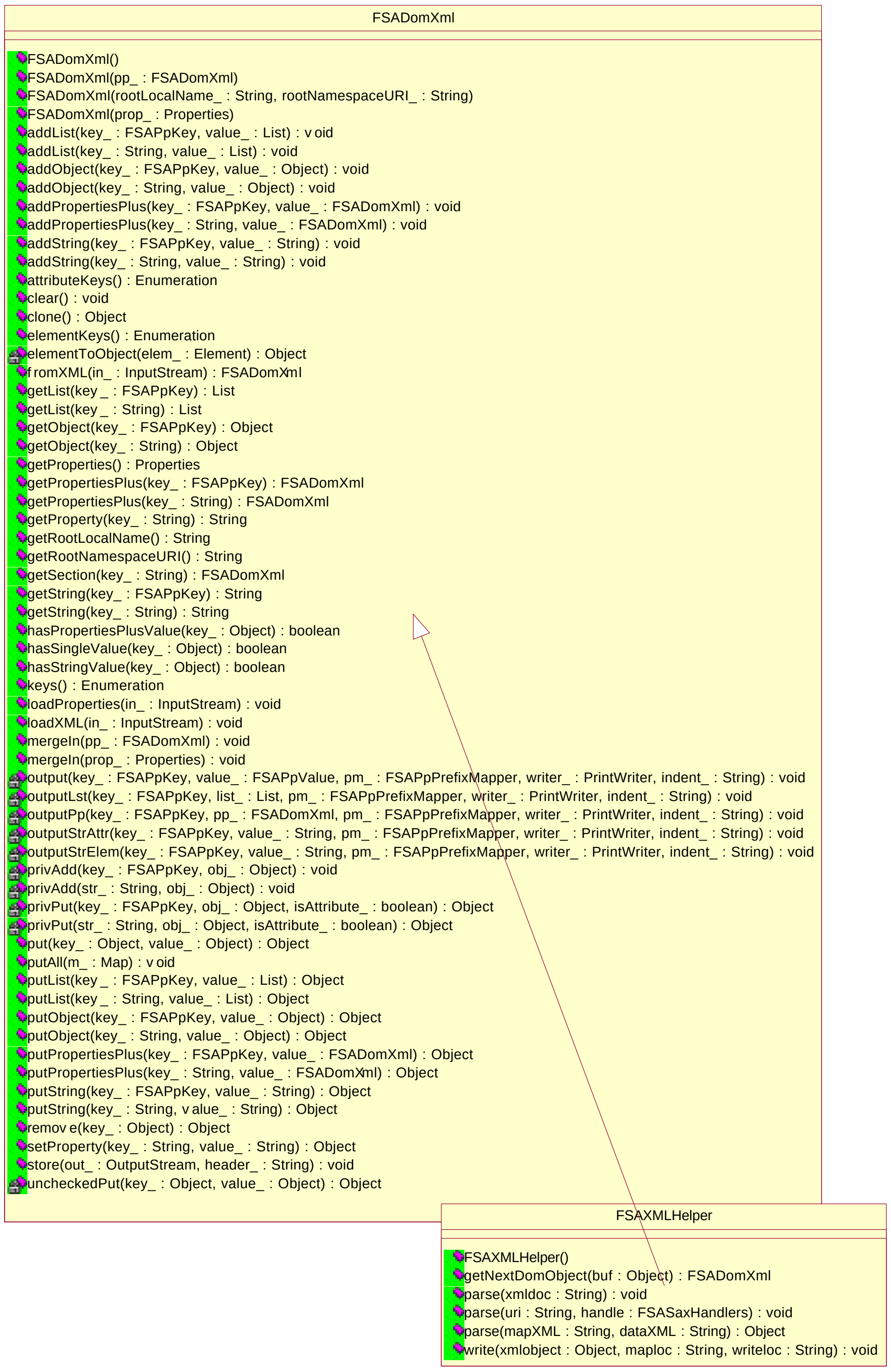
Methods	Arguments (Type, Name)	Valid Responses (Return Type, Exceptions Thrown)	Description
Public:			
parse	(String mapfile, String datafile)	Object Throws FSAException	Parses an XML document and returns an instance of a Java object that is defined by the mapping class.
write	(Object classfile,String mapfile, String writeloc)	Void Throws FSAException	Marshals an instance of the specified object to a XML Document

4.6.6 Class Diagrams

FSAXMLHelper


FSAXMLHelper
FSAXMLHelper()
getNextDomObject(buf : Object) : FSADomXml
parse(xmldoc : String) : void
parse(uri : String, handle : FSASaxHandlers) : void
parse(mapXML : String, dataXML : String) : Object
write(xmlobject : Object, maploc : String, writeloc : String) : void

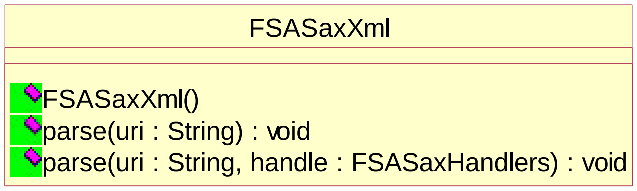
FSADomXml

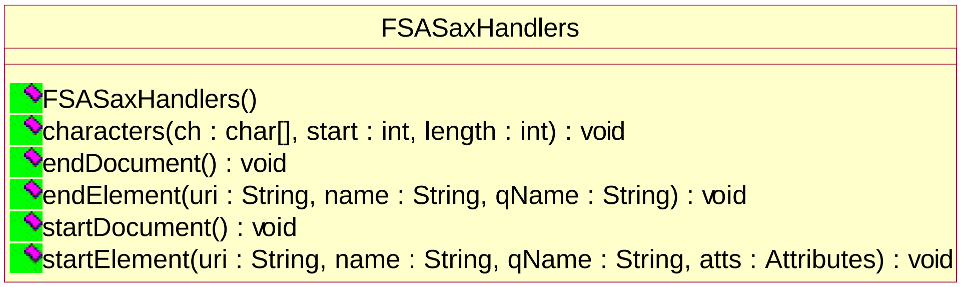


```
FSADomXml

FSADomXml()
FSADomXml(pp_ : FSADomXml)
FSADomXml(rootLocalName_ : String, rootNamespaceURI_ : String)
FSADomXml(prop_ : Properties)
addList(key_ : FSAPpKey, value_ : List) : v oid
addList(key_ : String, value_ : List) : void
addObject(key_ : FSAPpKey, value_ : Object) : void
addObject(key_ : String, value_ : Object) : void
addPropertiesPlus(key_ : FSAPpKey, value_ : FSADomXml) : void
addPropertiesPlus(key_ : String, value_ : FSADomXml) : void
addString(key_ : FSAPpKey, value_ : String) : void
addString(key_ : String, value_ : String) : void
attributeKeys() : Enumeration
clear() : void
clone() : Object
elementKeys() : Enumeration
elementToObject(elem_ : Element) : Object
f romXML(in_ : InputStream) : FSADomXml
getList(key_ : FSAPpKey) : List
getList(key_ : String) : List
getObject(key_ : FSAPpKey) : Object
getObject(key_ : String) : Object
getProperties() : Properties
getPropertiesPlus(key_ : FSAPpKey) : FSADomXml
getPropertiesPlus(key_ : String) : FSADomXml
getProperty(key_ : String) : String
getRootLocalName() : String
getRootNamespaceURI() : String
getSection(key_ : String) : FSADomXml
getString(key_ : FSAPpKey) : String
getString(key_ : String) : String
hasPropertiesPlusValue(key_ : Object) : boolean
hasSingleValue(key_ : Object) : boolean
hasStringValue(key_ : Object) : boolean
keys() : Enumeration
loadProperties(in_ : InputStream) : void
loadXML(in_ : InputStream) : void
mergeIn(pp_ : FSADomXml) : void
mergeIn(prop_ : Properties) : void
output(key_ : FSAPpKey, value_ : FSAPpValue, pm_ : FSAPpPrefixMapper, writer_ : PrintWriter, indent_ : String) : void
outputLst(key_ : FSAPpKey, list_ : List, pm_ : FSAPpPrefixMapper, writer_ : PrintWriter, indent_ : String) : void
outputPp(key_ : FSAPpKey, pp_ : FSADomXml, pm_ : FSAPpPrefixMapper, writer_ : PrintWriter, indent_ : String) : void
outputStrAttr(key_ : FSAPpKey, value_ : String, pm_ : FSAPpPrefixMapper, writer_ : PrintWriter, indent_ : String) : void
outputStrElem(key_ : FSAPpKey, value_ : String, pm_ : FSAPpPrefixMapper, writer_ : PrintWriter, indent_ : String) : void
privAdd(key_ : FSAPpKey, obj_ : Object) : void
privAdd(str_ : String, obj_ : Object) : void
privPut(key_ : FSAPpKey, obj_ : Object, isAttribute_ : boolean) : Object
privPut(str_ : String, obj_ : Object, isAttribute_ : boolean) : Object
put(key_ : Object, value_ : Object) : Object
putAll(m_ : Map) : v oid
putList(key_ : FSAPpKey, value_ : List) : Object
putList(key_ : String, value_ : List) : Object
putObject(key_ : FSAPpKey, value_ : Object) : Object
putObject(key_ : String, value_ : Object) : Object
putPropertiesPlus(key_ : FSAPpKey, value_ : FSADomXml) : Object
putPropertiesPlus(key_ : String, value_ : FSADomXml) : Object
putString(key_ : FSAPpKey, value_ : String) : Object
putString(key_ : String, v alue_ : String) : Object
remov e(key_ : Object) : Object
setProperty(key_ : String, value_ : String) : Object
store(out_ : OutputStream, header_ : String) : void
uncheckedPut(key_ : Object, value_ : Object) : Object
```

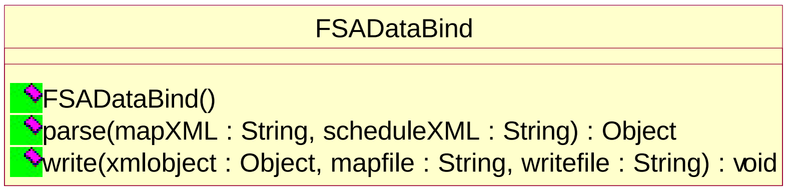
```
FSAXMLHelper

FSAXMLHelper()
getNextDomObject(buf : Object) : FSADomXml
parse(xmldoc : String) : void
parse(uri : String, handle : FSASaxHandlers) : void
parse(mapXML : String, dataXML : String) : Object
write(xmlobject : Object, maploc : String, writeloc : String) : void
```

FSASaxXml


FSASaxXml
FSASaxXml()
parse(uri : String) : void
parse(uri : String, handle : FSASaxHandlers) : void

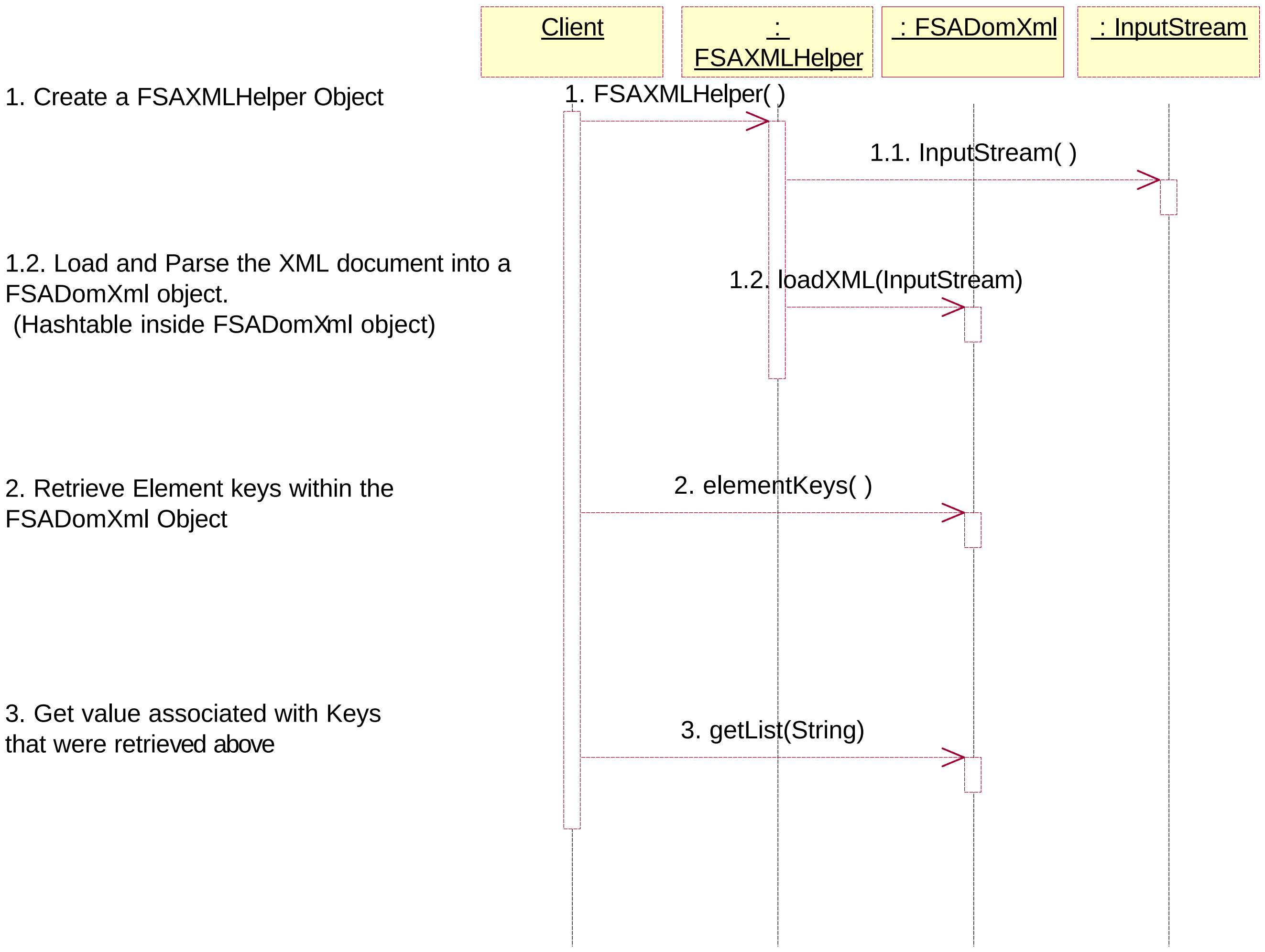

FSASaxHandlers
FSASaxHandlers()
characters(ch : char[], start : int, length : int) : void
endDocument() : void
endElement(uri : String, name : String, qName : String) : void
startDocument() : void
startElement(uri : String, name : String, qName : String, atts : Attributes) : void

FSADataBind

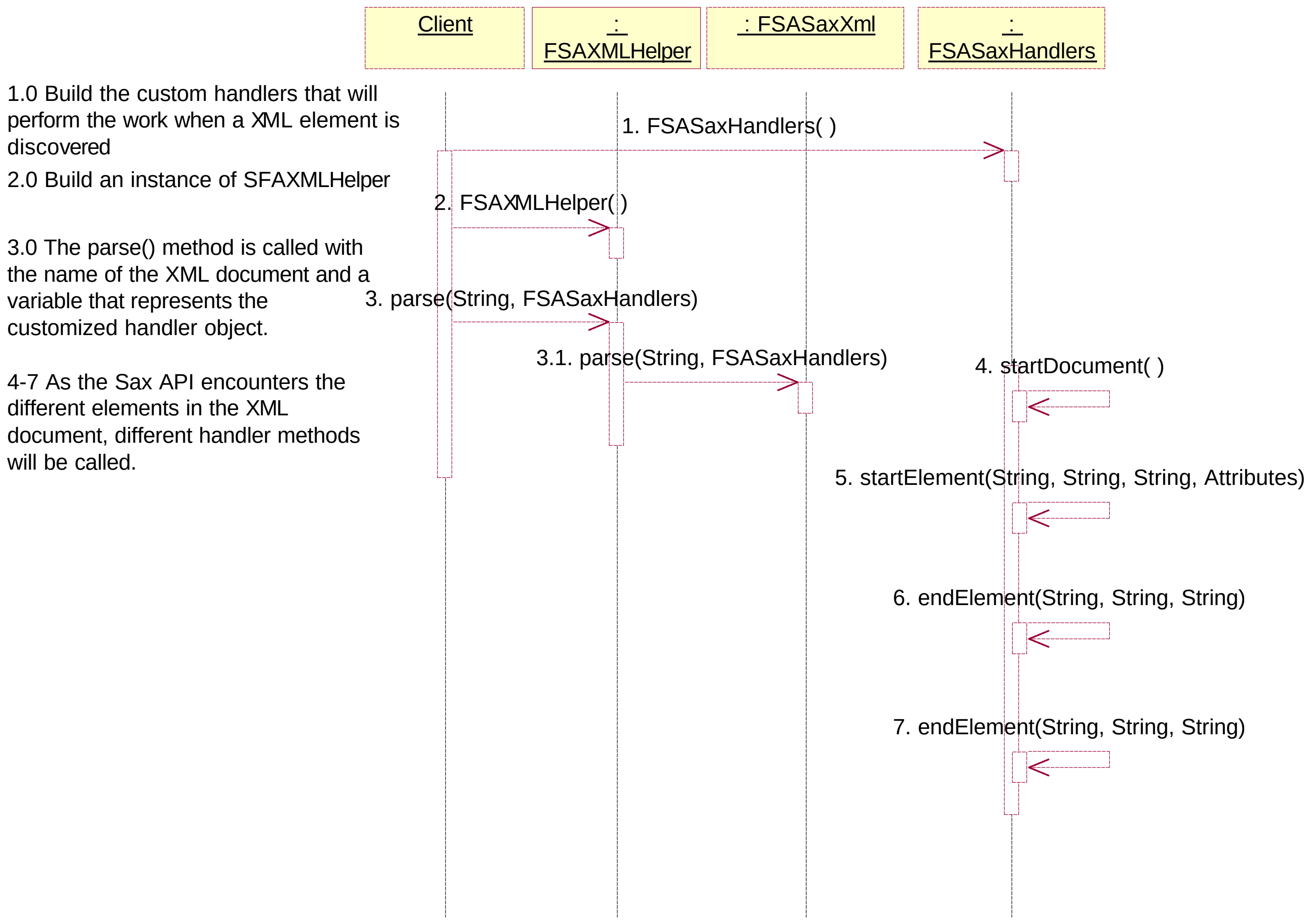

FSADataBind
FSADataBind()
parse(mapXML : String, scheduleXML : String) : Object
write(xmlobject : Object, mapfile : String, writefile : String) : void

4.6.7 Sequence Diagrams

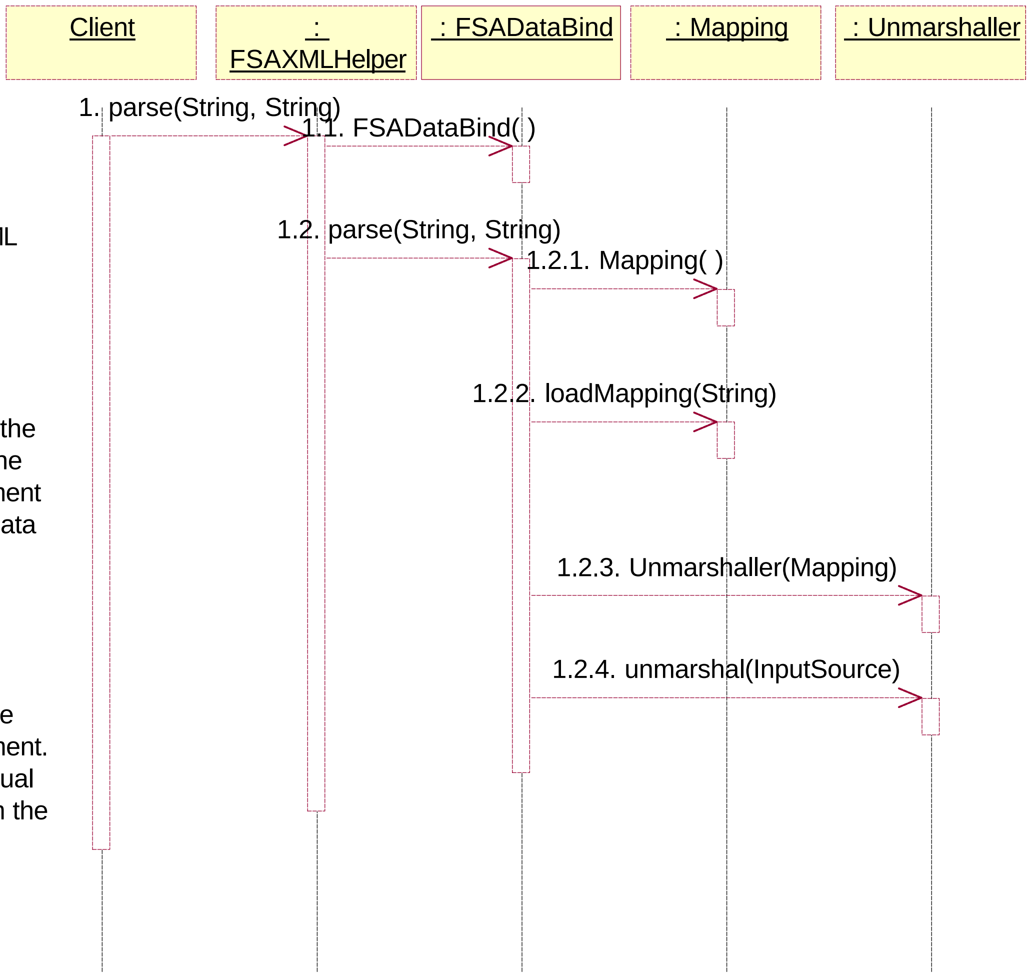
This sequence illustrates the interaction between a client object and FSADomXml object to parse an XML document using the DOM API.


Client
: FSAXMLHelper
: FSADomXml
: InputStream
1. Create a FSAXMLHelper Object
1. FSAXMLHelper( )
1.1. InputStream( )
1.2. Load and Parse the XML document into a FSADomXml object. (Hashtable inside FSADomXml object)
1.2. loadXML(InputStream)
2. Retrieve Element keys within the FSADomXml Object
2. elementKeys( )
3. Get value associated with Keys that were retrieved above
3. getList(String)

This sequence illustrates the interaction between a client object and FSASaxXml object to parse an XML document using the SAX API.


Client
: FSAXMLHelper
: FSASaxXml
: FSASaxHandlers
1.0 Build the custom handlers that will perform the work when a XML element is discovered
2.0 Build an instance of SFAXMLHelper
3.0 The parse() method is called with the name of the XML document and a variable that represents the customized handler object.
4-7 As the Sax API encounters the different elements in the XML document, different handler methods will be called.
1. FSASaxHandlers( )
2. FSAXMLHelper( )
3. parse(String, FSASaxHandlers)
3.1. parse(String, FSASaxHandlers)
4. startDocument( )
5. startElement(String, String, String, Attributes)
6. endElement(String, String, String)
7. endElement(String, String, String)

This sequence illustrates the interaction between a client object and FSADataBind object using a Field Descriptor XML document and a Class data XML document.



1.0 Client calls parse(String,String) with the Field descriptor XML document and the data XML document defined in the arguments

1.2 parse() builds a new mapping object that maps the field descriptor defined in the field descriptor XML document to the data defined in the data XML document.

1.2.2 - 1.2.4 The load Mapping() method loads the field descriptor XML document. The unmarshal method actual builds the Java Object from the data XML Document

4.6.8 References

- Castor XML Parsing Framework

  http://www.castor.org/

- XML Binding (JAXB)

  http://java.sun.com/xml/jaxb/

- World Wide Web Consortium

  http://www.w3.org/

- Sun Java web site

  http://java.sun.com/

- Sax Project

  http://www.saxproject.org/

4.7 JSP Tag Library

4.7.1 Purpose

The purpose of the ITA RCS JSP tag library framework is to provide a set of custom tags for developers to utilize to simplify, standardize, and extend the use of JSP tag libraries within the J2EE standard.

4.7.2 System Overview

The RCS JSP Tag Library framework is created using the Java programming language. The tag library framework provides a collection of commonly used JSP custom tag libraries for JSP developers to access. The JSP Tag Library framework is comprised of libraries leveraged from the Jakarta Struts framework, Apache Taglibs project, and custom developed libraries.

The following is a list of JSP tag libraries provided in this framework:

- **Jakarta Struts Bean Taglib** - contains custom JSP tags used to define new beans from a variety of sources and to render bean or bean property output response
- **Jakarta Struts HTML Taglib** - contains JSP custom tags useful in creating dynamic HTML user interfaces, including input forms
- **Jakarta Struts Logic Taglib** – contains tags useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management
- **Jakarta Struts Template Taglib** - contains tags that are useful in creating dynamic JSP templates for pages that share a common format
- **Jakarta DateTime Taglib** - contains tags that can be used to handle date and time related functions
- **Jakarta I18N Taglib** - contains tags that help manage the complexity of creating multi-lingual web applications
- **Jakarta Input Taglib** – contains tags that present HTML <form> elements tied to the ServletRequest. Can be used to pre-populate form elements with prior values that the user has chosen or with default values
- **Log Taglib** – is used to embed logging calls in JSP using the Logging Framework
- **Jakarta Page Taglib** – contains tags that can be used to access all of the PageContext information of a JSP page provided ‘page’attributes

- **Jakarta XSL Taglib** – contains tags used to process an XML document with an XSL stylesheet and incorporate the data in the page
- **Jakarta XTags Taglib** – contains custom tags for working with XML and implements an XSLT-like language allowing XML to be styled and processed from directly within a JSP. XTags is currently built on DOM4J foundation, an open source XML framework for the Java platform.

4.7.3 Design Considerations

4.7.3.1 Assumptions and Dependencies

The tag library framework functions in a J2EE application server environment. As the current production servers for FSA are running IBM's WAS v. 3.5.3 and v. 3.5.5, the framework will be compiled using its required JDK version 1.2.2. This framework is compatible with the current JavaServer Pages (1.1), Java Servlet (2.2), Struts (1.0.1), and Java Database Connectivity (2.0) specifications for this server. The tag library framework will be built and fully tested on the Sun Solaris 2.6 and HP-UX 11.0 operating systems. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.7.3.2 Goals and Guidelines

The goal of the JSP tag library framework is to provide a simple yet robust framework that may be utilized by FSA application teams conducting development in a Java environment. The tag library framework provides development teams with easy access to commonly used JSP custom tags.

Custom tags are extensions to the JSP language and are distributed in the form of a tag library. Tag libraries encourage a separation of duties between application developers and web designers, resulting in increased productivity by encapsulating recurring tasks as reusable components.

4.7.3.3 Development Methods

The JSP Custom Tag Library framework leverages many taglibs that have been developed by Apache. The tag libraries that will be developed by the ITA team will use generally used object-oriented programming practices.

4.7.4 Detailed Design

4.7.4.1 Struts – Bean Taglib (Package: org.apache.struts.taglib.bean)

The "struts-bean" tag library contains JSP custom tags useful to defining new beans (in any desired scope), from a variety of possible sources, as well as a tag to render a particular bean (or bean property) to the output response. Detailed information for this tag library can be found at:

http://jakarta.apache.org/struts/api-1.0/org/apache/struts/taglib/bean/package-summary.html.

Class	Description
CookieTag*[2]	Define a scripting variable based on the value(s) of the specified cookie received with this request.
Define Tag*	Define a scripting variable based on the value(s) of the specified bean property.
Header Tag*	Define a scripting variable based on the value(s) of the specified header received with this request.
IncludeTag*	Define the contents of a specified intra-application request as a page scope attribute of type String. If the current request is part of a session, the session id will be included in the generated request, so the request will be a part of the same session.
MessageTag	Retrieves an internationalized message string from the AcionResources object. This object is stored as a context attribute in the associated ActionServlet implementation.
PageTag*	Define a scripting variable that exposes the requested page context item both as a scripting variable and a page scope bean.
ParameterTag*	Define a scripting variable based on the value(s) of the specified parameter received with this request.
ResourceTag*	Define a scripting variable based on the contents of the specified web application resource.
SizeTag*	Define a scripting variable that will contain the number of elements found in a specified array, Collection, or Map.
StrutsTag*	Define a scripting variable that exposes the requested Struts internal configuration object.
WriteTag	Tag that retrieves the specified property of the specified bean, converts it to a String representation (if necessary), and writes it to the current output stream, optionally filtering characters that are sensitive in HTML.

4.7.4.2 Struts – HTML Taglib (Package: org.apache.struts.taglib.html)

The "struts-html" tag library contains JSP custom tags for creating dynamic HTML user interfaces, including input forms. Detailed information for this tag library can be found at: http://jakarta.apache.org/struts/api-1.0/org/apache/struts/taglib/html/package-summary.html.

Class	Description
BaseFieldTag	Convenience base class for the various input tags for text fields.
BaseHandlerTag	Base class for tags that render form elements capable of including JavaScript even handlers and/or CSS Style attributes.
BaseInputTag	Abstract base class for the various input tags.
BaseTag	Renders an HTML element with an href attribute pointing to the absolute location of the enclosing JSP page. This tag is only valid when nested inside a head tag body. The presence of this tag allows the browser to resolve relative URL's to images, CSS stylesheets

[2] Note: ***** next to a tag indicates that the TagExtraInfo (TEI) implementation of the class is available (i.e.. CookieTag* means that there is a CookieTei. All TEI classes contain the method getVariableInfo, which returns information about the scripting variable to be created).

	and other resources in a manner independent of the URL used to call the ActionServlet. There are no attributes associated with this tag.
ButtonTag	Convenience base class for the various input tags for text fields.
CancelTag	Tag for input fields of type "cancel".
CheckboxTag	Tag for input fields of type "checkbox".
Constants	Manifest constants for this package.
ErrorsTag	Tag that renders error messages if an appropriate request attribute has been created.
FileTag	Tag for input fields of type "file".
FormTag	Tag that represents an input form, associated with a bean whose properties correspond to the various fields of the form.
HiddenTag	Tag for input fields of type "text".
HtmlTag	Renders an HTML element with appropriate language attributes if there is a current Locale available in the user's session.
ImageTag	Tag for input fields of type "image".
ImgTag	Generate an IMG tag to the specified image URI.
LinkTag	Generate a URL-encoded hyperlink to the specified URI.
MultiboxTag	Tag for input fields of type "checkbox".
OptionsTag	Tag for creating multiple <select> options from a collection.
OptionTag	Tag for select options.
PasswordTag	Tag for input fields of type "text".
RadioTag	Tag for input fields of type "radio".
ResetTag	Tag for input fields of type "reset".
RewriteTag	Generate a URL-encoded URI as a string.
SelectTag	Tag that represents an HTML select element, associated with a bean property specified by our attributes.
SubmitTag	Tag for input fields of type "submit".
TextareaTag	Tag for input fields of type "textarea".
TextTag	Tag for input fields of type "text".

4.7.4.3 Struts – Logic Taglib (Package: org.apache.struts.taglib.logic)

The "struts-logic" tag library contains tags that for managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management. Detailed information for this tag library can be found at: http://jakarta.apache.org/struts/api-1.0/org/apache/struts/taglib/logic/package-summary.html

Class	Description
CompareTagBase	Abstract base class for comparison tags.
ConditionalTagBase	Abstract base class for the various conditional evaluation tags.
EqualTag	Evaluate the nested body content of this tag if the specified variable and value are equal.
ForwardTag	Perform a forward or redirect to a page that is looked up in the global ActionForwards collection associated with the application.
GreaterEqualTag	Evaluate the nested body content of this tag if the specified variable is greater than or equal to the specified value.
GreaterThanTag	Evaluate the nested body content of this tag if the specified variable is greater than the specified value.
IterateTag*	Custom tag that iterates the elements of a collection, which can be either an attribute or the property of an attribute.
LessEqualTag	Evaluate the nested body content of this tag if the specified variable is less than the specified value.
LessThanTag	Evaluate the nested body content of this tag if the specified variable is less than the specified value.
MatchTag	Evaluate the nested body content of this tag if the specified value is a substring of the specified variable.
NotEqualTag	Evaluate the nested body content of this tag if the specified variable and value are not equal.
NotMatchTag	Evaluate the nested body content of this tag if the specified value is not a substring of the specified variable.
NotPresentTag	Evaluate the nested body content of this tag if the specified value is not present for this request.
PresentTag	Evaluate the nested body content of this tag if the specified value is present for this request.
RedirectTag	Generate a URL-encoded redirect to the specified URI.

4.7.4.4 Struts – Template Taglib (Package: org.apache.struts.taglib.template)

The "struts-template" tag library contains tags that are useful for creating dynamic JSP templates for pages that share a common format. Detailed information for this tag library can be found at:
http://jakarta.apache.org/struts/api-1.0/org/apache/struts/taglib/template/package-summary.html

Class	Description
GetTag	Tag handler for <template:get>, which gets content from the request scope and either includes the content or prints it, depending upon the value of the content's direct attribute.
InsertTag	Tag handler for <template:insert>, which includes a template.
PutTag	Tag handler for <template:put>, which puts content into the request scope.

4.7.4.5 Date Taglib (Package: org.apache.taglibs.datetime)

The DateTime custom tag library contains tags, which can be used to handle date, and time related functions. Tags are provided for formatting a Date for output, generating a Date from HTML form input, using time zones, and localization. This is the Jakarta Page Taglib Version 1.0 – Beta 1.

Class	Description
AmPmsTag*	Loops through all the am / pm strings so that they can be accessed by using the standard JSP <jsp:getProperty> tag.
CurrentTimeTag	Obtains the current time in milliseconds since Jan 1, 1970 GMT.
ErasTag*	Loops through all the era (i.e., period of time) strings so that they can be accessed by using the standard JSP <jsp:getProperty> tag.
FormatTag	Format a Date for display.
MonthsTag*	Loops through all the months of the year so that month names can be accessed by using the standard JSP <jsp:getProperty> tag.
ParseTag	Parses a Date string and output the time in ms.
TimeZonesTag*	Loop through all the TimeZone's so that ID's and Display Names can be accessed by using the standard JSP <jsp:getProperty> tag.
TimeZoneTag	Sets the client TimeZone for the session as a script variable.
WeekdaysTag*	Loops through the days of the week so that weekday names can be accessed by using the standard JSP <jsp:getProperty> tag.

4.7.4.6 I18N Taglib (Package: org.apache.taglibs.i18n)

The i18n custom tag library contains tags that help manage the complexity of creating multi-lingual web applications. These tags provide similar (though not identical) functionality to the internationalization available in the Struts framework, but do not require adopting the entire Struts framework.

Class	Description
BundleTag*	Implements an empty tag that allows developers to use a resource bundle to internationalize content in a web page.
ConditionalTagSupport	Provides the base implementation for the ifdef and ifndef tags.
FormatCurrencyTag	Formats a Number instance using the current Locale and the Currency NumberFormat.
FormatDateTag*	Formats a Date instance using a Locale and either a DateFormat or a pattern based SimpleDateFormat.
FormatDateTagSupport	Abstract base class which supports the defaulting of the value to 'now' if no other value is specified

	is specified.
FormatDateTimeTag*	Formats a Date instance using the default Date and Time formatter for the current Locale.
FormatNumberTag*	Formats a Number instance using a Locale and the NumberFormat or a DecimalFormat if a pattern is specified.
FormatPercentTag*	Formats a Number instance using the current Locale and the Percent NumberFormat.
FormatStringTag*	A simple tag that allows a String to be output with null handling.
FormatTagSupport	An abstract base class for the formatting tags to provide implementation inheritance.
FormatTimeTag*	Formats a Date instance using a Locale and the default time format.
IfdefTag	This class implements a body tag that allows developers to use a resource bundle to internationalize content in a web page. The ifdef tag allows the JSP author to conditionally evaluate sections of a JSP based on whether a value is provided for the given key.
IfndefTag	Implements body tag that allows you to use a resource bundle to internationalize content in a web page. The ifndef tag allows the JSP author to conditionally evaluate sections of a JSP if a value is not provided for the given key.
LocaleTag*	Defines a Locale context for use by other inner JSP tags.
MessageArgumentsTag	Utilized inside a MessageTag to create an ordered list of arguments to use with java.text.MessageFormat.
MessageTag*	Implements a body tag that allows you to use a resource bundle to internationalize content in a web page.
ResourceHelper	Used by the locale and message tags for caching the ResourceBundle in the session and request.

4.7.4.7 Input Taglib (Package: org.apache.taglibs.input)

The input tag extension library features the presentment of HTML <form> elements that are tied to the ServletRequest calling the JSP page. Forms elements can be pre-populated with prior values that the user has chosen -- or with default values for the first time user of a web page. This is useful when the same page needs to be presented to the user several times. Server-side validation is a good example of this process.

It is also possible to automatically build up <select> boxes, making it easier to build data-driven forms. Even if the same page is presented multiple times, and the form elements that have default values are desired, this library provides this functionality to free programmers from writing extensive code.

Class	Description
Checkbox	Implements the <input:checkbox> tag, which presents a check box form element.
Radio	Implements the <input:radio> tag, which presents a radio button form element.
Select	Implements the <input:select> tag, which presents a select form element.
Text	Implements the <input:text> tag, which presents a text form element.
TextArea	Implements the <input:textarea> tag, which presents a <textarea> form element.

4.7.4.8 Log Taglib (Package: gov.ed.fsa.ita.taglibs.log)

The Log library allows embedding logging calls in JSP using the ITA RCS logging framework. This tag library is leveraged from the Jakarta Log tag extension library except it uses the ITA RCSlogging framework instead of the log4j project.

Class	Description
LogTag	Logs a message to the current logging category (i.e., debug) using the RCS logging framework.

4.7.4.9 Page Taglib (Package: org.apache.taglibs.page)

Used to access all of the information about the PageContext of a JSP page. This is the Jakarta Page Taglib Version 1.0 – Beta 1. This tag library uses the <jsp:getProperty> tag that is not supported by JSP 1.1, this implies that custom development may be required, therefore, leveraging the sample provided by Jakarta may be necessary. Sample code is available on the Jakarta website at: http://jakarta.apache.org/taglibs/doc/page-doc

Class	Description
AttributeTag	Used to output the value for a single PageContext attribute named with **name**.
AttributesTag*	Accesses all of the PageContext information for a JSP page from the 'page' scoped attributes. Loops through all of the attributes.
EqualsAttributeTag	Determines if a PageContext attribute equals the value of the "match" tag attribute. Includes the body of the tag if the attribute equals the value of the "match" tag attribute.
ExistsAttributeTag	Determine if a PageContext attribute exists. Includes the body of the tag if the attribute exists.
RemoveAttributeTag	Removes a PageContext attribute with specified name from the body of the tag.
SetAttributeTag	Sets a PageContext attribute name to a String from the body of the tag.

4.7.4.10 XSL Taglib (Package: org.apache.taglibs.xsl)

With this custom tag library it is possible to process an XML document with an XSL stylesheet and insert it in the document. XSL tag library acts as an example of custom tag library code techniques, rather than actual production tags.

Interface	Description
XSLProcessor	This interface is implemented by the different XSL processes to ensure the process method is called.

Class	Description
ApplyTag	Apply an XSL stylesheet to an XML data source, rendering the output to the writer of the JSP page. This tag uses the Xalan XSLT processor.
ExportTag	Export the content of the specified JSP bean (in the specified scope) to the output writer, presumably after modifications have been completed.
ImportTag	Import the content of the specified page, and assign it (as a String) to the specified scripting variable in the specified scope.
IncludeTag	Include the contents of the specified page at this point in the output. This tag is similar to jsp:include, but does not cause the output to be sent directly to the servlet response. Therefore, it can be used to capture the content of the page as body content of a surrounding tag in which we are nested.
ShowSource	Display the sources of the JSP file.
XalanXSLProcessor	Xalan XSL Processor.
XSLProcessorFactory	XSL processor factory.

4.7.4.11 XTags Taglib

XTags is a JSP custom tag library for working with XML. XTags implements an XSLT-like language allowing XML to be styled and processed directly within a JSP page using familiar XSLT and XPath techniques. XTags is similar to XSLT but implemented in JSP. XTags allows the developer to work with multiple JSPs, custom tags, within a single JSP. XTags is currently built on a DOM4J foundation (http://DOM4J.org).

4.7.4.11.1 XTags – Servlet (org.apache.taglibs.xtags.servlet)

Contains the XPathServlet, which outputs the value of an XPath expression of an XML document.

Class	Description
XpathServlet	A Servlet to display the result of an XPath expression as XML

4.7.4.11.2 XTags – Util (org.apache.taglibs.xtags.util)

This tag library contains a collection of collections, utility and helper classes.

Class	Description
JspHelper	A collection of helper methods for JSP tag implementors.
JspVariableContext	A Servlet to display the result of an XPath expression as XML.
StringHelper	A Helper class of useful String methods.
URLHelper	Helper methods for creating URLs that can handle relative or absolute URIs or full URLs.

4.7.4.11.3 XTags – XPath (org.apache.taglibs.xtags.xpath)

JSP Custom tags for working with XML via XPath for navigating, processing and transforming XML documents using XPath expressions, navigation, iteration and XSLT stylesheets.

Interface	Description
ContextNodeTag	An abstract base class that represents a context node on which XPath expressions can be evaluated.

Class	Description
AbstractBodyTag	Abstract base class for BodyTag implementations.
AbstractTag	A tag that performs an XPath expression on the current context Node.
AddTag	The add tag parses it's body (as an XML fragment) and appends the contents to the current node.
ApplyTemplatesTag	The body of this tag defines a stylesheet that is implemented via calling a JSP include.
AttributesTag	Adds an XML attribute to the parent element tag like the <xsl:attribute> tag.
BodyAction	An Action that tells the Stylesheet tag which template body to execute.
BreakTag	Causes the current iteration to be terminated rather like the Java 'break' statement.
ChooseTag	Behaves like the equivalent XSLT tag, <xsl: Choose> is used to filter out conditions of an XML document
ContextTag	Changes the current context for tags used inside its body.
CopyOfTag	Performs a copy-of operation like the XSLT tag.
CopyTag	Performs a copy operation like the XSLT tag - a shallow copy.
ElementTag	Produce an XML element that can contain other attributes or elements like the <xsl:element> tag.
ForEachTag	Performs iteration over the results of an XPath expression on an XML document.
ForEachTagExtraInfo	The extra info for each tag.

IfTag	Behaves like the equivalent XSLT tag.
JspAction	An Action that includes a piece of JSP.
JspCopyOfAction	Outputs the given Node to the current JSP output.
JspValueOfAction	Outputs the string-value of the given Node (as defined by the XPath specification) to the current JSP output.
OtherwiseTag	Behaves like the equivalent XSLT tag.
OutputTag	Specifies the output format of the XML.
ParseTag	Parses its body as an XML Document and defines a variable.
ParseTagExtraInfo	The extra info for the XML tag.
ReflectionAction	Calls a void method on an instance when the action is initiated.
RemoveTag	Removes nodes from the current document that matches the given XPath expression.
ReplaceTag	Parses it's body (as an XML fragment) and replaces the contents to the current node with this new XML fragment.
StylesheetTag	The body of this tag defines a stylesheet that is implemented via calling a JSP include.
StylesheetValueOfAction	Creates the string-value of the given Node (as defined by the XPath specification) and passes it to the parent StylesheetTag for output at the correct time.
TagHelper	A number of helper methods.
TemplateTag	The body of this tag defines a stylesheet that is implemented via calling a JSP include.
ValueOfTag	Performs an XPath expression on the current context Node.
VariableTag	A tag, which defines a variable from an XPath expression.
VariableTagExtraInfo	The extra info for the variable tag.
WhenTag	Behaves like the equivalent XSLT tag.

4.7.4.11.4 XTags – XSLT (org.apache.taglibs.xtags.xslt)

JSP Custom tags for working with XSLT.

Interface	Description
ParameterAcceptingTag	A tag that is capable of accepting a parameter value.

Class	Description
ParamTag	A tag that declares a parameter for use in an XSLT stylesheet.
StyleTag	A tag that performs an XSLT transformation on a given XML document.

4.7.5 References

- The Jakarta Taglibs Project

  http://jakarta.apache.org/taglibs/

- Core Servlets and JavaServer Pages – Chapter 14: Creating Custom Tag Libraries

  http://developer.java.sun.com/developer/Books/javaserverpages/cservletsjsp/chapter14.pdf

- The Struts Framework Project

  http://jakarta.apache.org/struts

- Struts Framework API (Version 1.0)[3]

  http://jakarta.apache.org/struts/api-1.0/index.html

- XTags is built on DOM4J

  http://DOM4J.org

[3] Version 1.0.1 is a patch release for version 1.0.

4.8 Scheduler

4.8.1 Purpose

In the past FSA applications have had different scheduling needs, which had been accomplished using different methods. FSA has identified a need to launch email and FTP programs within the Java Virtual Machine (JVM) environment at specific times once or several times a month to broadcast data or messages. In addition, there is a need to launch database maintenance programs from within Java classes at specific times to retrieve data out of databases.

To accomplish these tasks, FSA has had to rely on Unix commands to launch Unix scripts that integrate the different functionalities with the Java Virtual Machine. This process has been clumsy and unreliable. The FSA Java Scheduler allows developers to set up scheduled launches of Java classes at specified times, just like Unix cron. In fact the syntax was purposely developed to be similar to Unix cron so that administrators and developers would be familiar with it.

As with all RCS components, a common RCS logging and exception handling framework is provided that enhances the reliability and serviceability of the particular framework. These frameworks have a well-documented test and performance strategy that ensures their availability.

4.8.2 System Overview

The ability to schedule a task is critical to most Java enterprise applications. In most cases, Java business applications have a need to schedule a batch task during periods of low use or maintenance. To accomplish this, most Java developers have had to rely on techniques outside of the Java Virtual Machine programming paradigm to achieve stated goals. This forces the developer to rely on API's that the Java engine has no control over. Java cannot typically respond to whether the task was accomplished or not, without significant code enhancements. These requirements bring about a need for a reliable Scheduling framework that is easy to implement. The FSA Scheduling framework is composed of an open-source timing framework called JDring, as well as, other RCS frameworks that provide Logging, Exception handling, and XML document parsing – that serve as a layered foundation to RCS customization.

JDring is an open source Java package, which provides an alarm scheduling system. JDring is well known among the open source developers and comes highly recommended for scheduling tasks.

The scheduling framework is built on these open source frameworks and provides the FSA developer with the ability to program a scheduled task or schedule a task via an XML document that can be reread anytime.

4.8.3 Design Considerations

4.8.3.1 Assumptions and Dependencies

JDring v1.3.1 must be used with the Java Development kit 1.2.2, as it relies on the Java Collections package provided with the JDK 1.2.2. This framework will function in a J2EE application server environment. As the current production server for FSA is IBM WebSphere 3.5.x, the framework will be compiled using its required JDK version 1.2.2. Compatibility is assumed with the current JavaServer Pages (1.1), Java Servlet (2.2), Java Messaging Service (1.0.1), and Java Database Connectivity (2.0) specifications for this server. This framework is fully tested on Sun Solaris 2.6 and HP-UX 11.0. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.8.3.2 Goals and Guidelines

The goal of this development is to provide a simple and robust Scheduling Framework that can be leveraged by FSA application teams developing applications in a Java environment, or more specifically WebSphere. It is critical that the ITA Scheduling Framework be reliable and easy to debug if problems occur. To this end the RCS Logging and Exception Handling Framework will be used to document any errors that the Scheduling Framework encounters.

Another goal of this framework is to standardize FSA applications on a single Scheduling method that allows all Java Based applications use to schedule tasks. This helps with support and maintenance of FSA applications.

A final goal is to ensure that the performance of the Scheduler Framework does not hinder the performance of the application using it. Part of the release process of RCS Frameworks is to ensure that performance is not an issue. This is accomplished through performance testing and documenting performance results.

4.8.3.3 Development Methods

Development of this framework relies on general object-oriented software development techniques, which are specified in any standard text on the Java programming language. Given the facility of its class and relationship patterns, no object oriented modeling tool or methodology was specifically used in its design. However, the resulting class files have been documented with standard class diagrams and sequence diagrams using Rational Rose in order to illustrate its structure more readily. These diagrams will prove useful to developers unfamiliar with this framework.

4.8.4 System Architecture

The ITA Scheduler Framework has been designed to manage a large quantity of scheduled tasks and alarms. The ITA Scheduler Framework is intended to trigger events when alarms' date and time match the current ones. Alarms are added dynamically in any order and can be one time or repetitive (i.e., rescheduled when matched). Two FSA Scheduler classes have been defined to incorporate the FSA Scheduler Framework. They are FSASchedule and FSAScheduleEvent.

4.8.4.1 Subsystem Architecture: FSASchedule

FSASchedule is a class that inherits from and adds functionality to the JDring scheduler framework. The FSASchedule class has the following features:

- Ability to add alarms via inline code that will launch tasks at specified dates and times. The AddAlarm methods support passing a java.util.date argument, cron like argument (min, day of week, day of month, month, year), an integer argument (number of seconds to delay) or a FSAScheduleEntry argument, which is used for XML parsing

- Ability to add alarms via XML documentation that will launch tasks at specified dates and times

- Boolean function that will check if an alarm exists within a specific FSASchedule object

- Ability to retrieve and inspect all alarms set

- Ability to delete a particular alarm or all alarms

4.8.4.2 Subsystem Architecture: FSAScheduleEntry

The FSAScheduleEntry class is used in partnership with the RCS XMLHelper framework to accomplish the XML document parsing. Using the XMLHelper framework, a developer can instantiate an instance of the FSAScheduleEntry class from an XML document and pass it to the FSASchedule class via the addAlarms methods.

4.8.4.3 Subsystem Architecture: ITA RCS XMLHelper Framework

The XMLHelper framework is an XML document-parsing framework that allows developers to use DOM, SAX or JAXB-like parsing. This framework has been added to the scheduler framework to allow the framework to read in new scheduling and data parameters from an XML document.

4.8.4.4 SubSystem Architecture: ITA RCS Logging and Exception Handling Framework

The ITA Logging and Exception Handling frameworks have been added to the scheduler framework to enhance debugging and tracing abilities. This will enable the capability to isolate problems that may occur within the framework.

4.8.5 Detailed System Design

4.8.5.1 FSASchedule

The FSASchedule class is a wrapper class that inherits directly from the AlarmManager class of the JDring Framework. The FSASchedule class provides simplified interfaces to the JDring framework as well as adding XML parsing capabilities. The FSASchedule class accomplishes the scheduling functionality by setting up an event listener. This event listener has an action associated with it. When the time / date arrives, then the action is executed. The FSASchedule class has the capability to launch any public method of another class in the scheduled future, as long as that class is in the current Classpath.

4.8.5.1.1 FSASchedule

Class Name:	FSASchedule
Component:	Scheduler
Description:	Public class
Package:	gov.ed. fsa.ita.schedule
Superclass:	fr.dyade.jdring.AlarmManager

Con/Destructors	**Arguments** (Type, Name)	**Description**
FSASchedule	none	Constructor

Methods	**Arguments** (Type, Name)	**Valid Responses** (Return Type, Exceptions Thrown)	**Description**
Public:			
addAlarm	(FSAScheduleEntry E)	FSAScheduleEntry throws FSAException	This method takes a FSAScheduleEntry object and adds the alarm to the schedule.
configureXML	(String, string)	FSAScheduleEntry throws FSAException	This method takes two string arguments. The first argument specifies the translation class-mapping file for the XML parser. The second argument specifies the file that holds actual class element values that will be marshaled. The purpose of the method is to instantiate FSASchedule from an XML document.
containsAlarm	(FSAScheduleEntry E)	boolean	This method takes a FSAScheduleEntry object and checks to see if that alarm is present in the current FSASchedule object.
getAllAlarms	none	java.util.List	This method returns a List of all scheduled alarms.
launchMethod	(FSAScheduleEntry E)	FSAScheduleEntry throws FSAException	This method takes a FSAScheduleEntry object and launches the specified class and method.
removeAllAlarms	none	void	This method removes all specified alarms within a FSASchedule Object.
removeAlarm	(FSAScheduleEntry E)	boolean	This method takes a FSAScheduleEntry object and removes the specified alarm.

4.8.5.2 FSAScheduleEvent

The RCS XMLHelper parsing framework is used to marshall almost any bean-like Java object to and from XML format. The FSASchedule class uses FSAXMLHelper to parse two documents (mapping.xml and schedule.xml) to instantiate an instance of FSAScheduleEvent. The mapping.xml document is used to define class descriptors and field descriptors of the FSAScheduleEntry to the FSASchedule framework. In other words, the attributes of the variables that are to be used by FSAScheduleEntry are defined within the mapping.xml document.

The mapping.xml document will define if the FSAScheduleEntry variables are loaded directly from the values defined in schedule.xml or the values get set via methods. The mapping.xml document should not have to be modified by the users of the FSASchedule framework. The Schedule XML document is the primary document that developers use to pass alarm entries into the Schedule Framework.

The Schedule.XML document holds the values of the variables used to fill out an FSAScheduleEntry class. The FSAScheduleEvent class is a Java Data Bean that represents the XML documents that were parsed by the FSASchedule configureXML method. Once the FSASchedule framework parses the XML document schedule.xml, and instantiates an FSAScheduleEntry class, the framework schedules the alarm with FSASchedule.

4.8.5.2.1 FSAScheduleEntry

Class Name:	FSAScheduleEntry
Component:	Scheduler
Description:	Public class
Package:	gov.ed. fsa.ita.schedule
Superclass:	Object

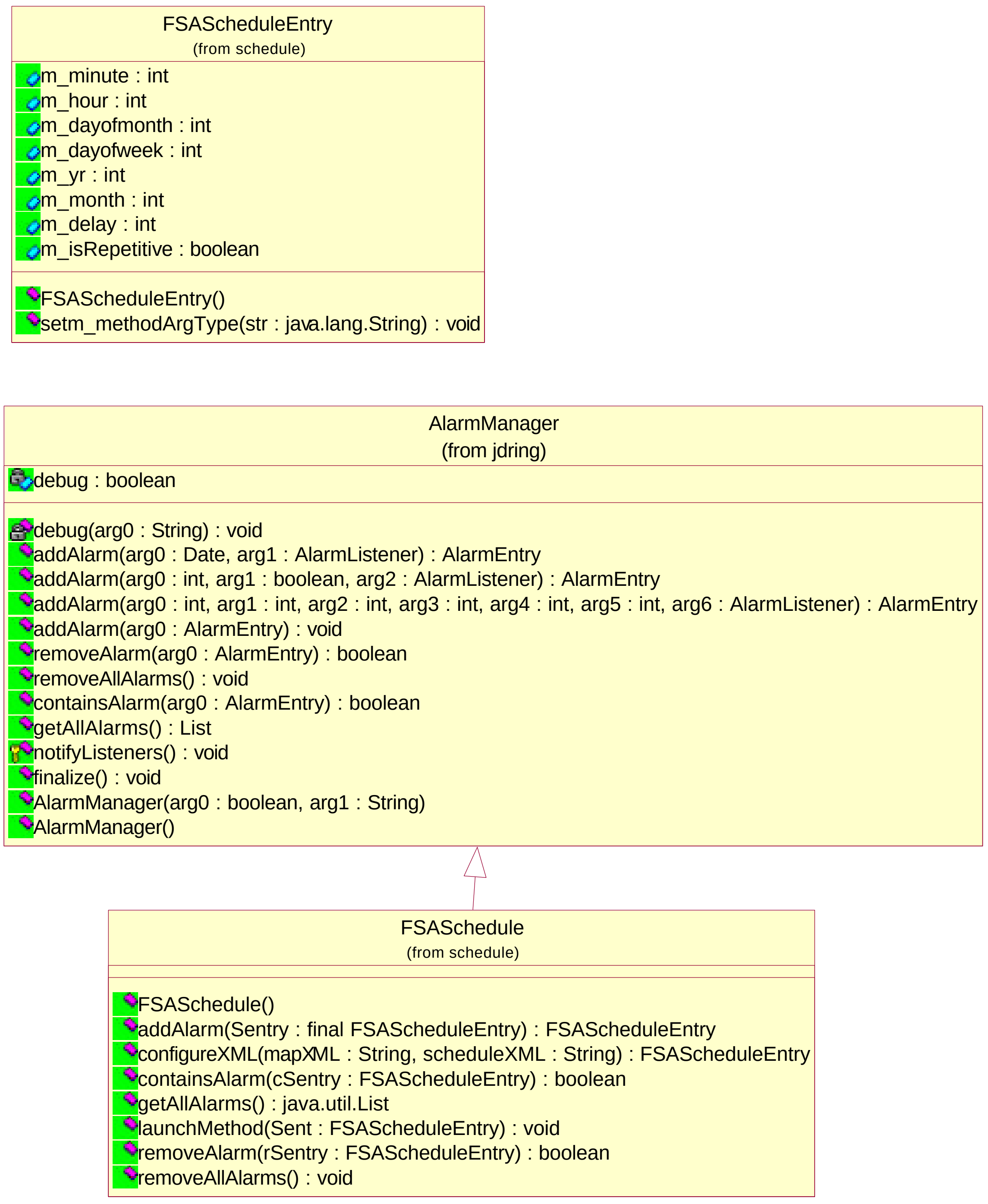
Con/Destructors	**Arguments** (Type, Name)	**Description**
FSAScheduleEntry	none	Constructor

Attribute	**Type**	**Description**
Public:		
m_classname	java.lang.String	String variable to specify name of the class Object.
m_dayofmonth	int	Integer variable that is used to specify day of month.
m_dayofweek	int	Integer variable that is used to specify day of week.
m_delay	int	Integer variable that is used to delay before alarm is signaled.
m_hour	int	Integer variable that is used to specify hour.
m_isRepetitive	boolean	Boolean variable that is used to specify whether delay should be repeated.
m_methodArgs	Object[]	Object array that will hold method arguments.

Attribute	Type	Description
m_methodArgType	Class[]	Class array that will hold method arguments types.
m_methodname	java.lang.String	String that will hold method name.
m_minute	int	Integer variable that is used to specify minute.
m_month	int	Integer variable that is used to specify month.
m_yr	int	Integer variable that is used to specify year.
postedAlarm	fr.dyade.jdring.AlarmEntry	AlarmEntry variable that is super.

4.8.6 Class Diagrams

FSASchedule Class


FSAScheduleEntry
(from schedule)
m_minute : int
m_hour : int
m_dayofmonth : int
m_dayofweek : int
m_yr : int
m_month : int
m_delay : int
m_isRepetitive : boolean
FSAScheduleEntry()
setm_methodArgType(str : java.lang.String) : void
AlarmManager
(from jdring)
debug : boolean
debug(arg0 : String) : void
addAlarm(arg0 : Date, arg1 : AlarmListener) : AlarmEntry
addAlarm(arg0 : int, arg1 : boolean, arg2 : AlarmListener) : AlarmEntry
addAlarm(arg0 : int, arg1 : int, arg2 : int, arg3 : int, arg4 : int, arg5 : int, arg6 : AlarmListener) : AlarmEntry
addAlarm(arg0 : AlarmEntry) : void
removeAlarm(arg0 : AlarmEntry) : boolean
removeAllAlarms() : void
containsAlarm(arg0 : AlarmEntry) : boolean
getAllAlarms() : List
notifyListeners() : void
finalize() : void
AlarmManager(arg0 : boolean, arg1 : String)
AlarmManager()
FSASchedule
(from schedule)
FSASchedule()
addAlarm(Sentry : final FSAScheduleEntry) : FSAScheduleEntry
configureXML(mapXML : String, scheduleXML : String) : FSAScheduleEntry
containsAlarm(cSentry : FSAScheduleEntry) : boolean
getAllAlarms() : java.util.List
launchMethod(Sent : FSAScheduleEntry) : void
removeAlarm(rSentry : FSAScheduleEntry) : boolean
removeAllAlarms() : void

4.8.7 Sequence Diagrams

This sequence illustrates the interaction between a client object and FSASchedule object to parse an XML document and set a scheduled entry.

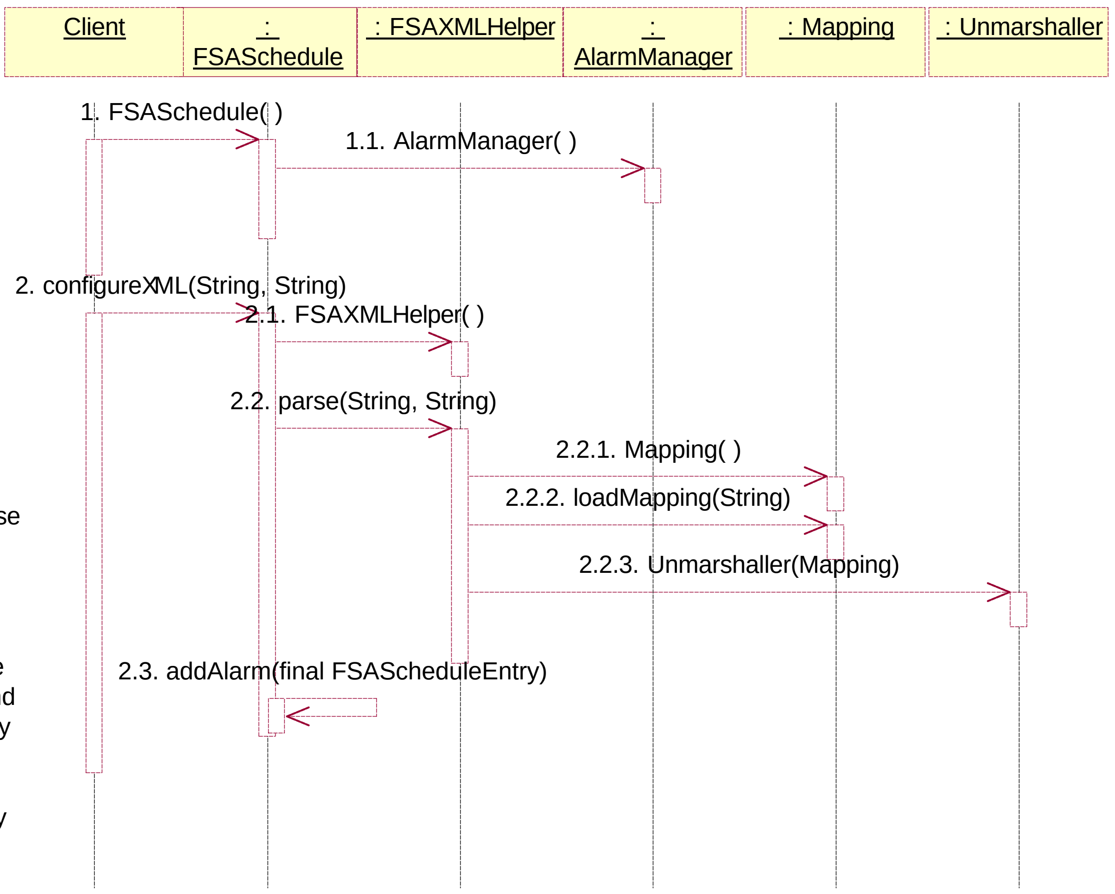
1.0 Build new instance of FSASchedule

2.0 Call configureXML with the location of the mapping xml document and the schedule xml document

2.2 configureXML calls parse and builds a new mapping object and then loads the mapping xml document.

The unmarshaller reads the schedule xml document and builds a FSAScheduleEntry object

2.3 The FSAScheduleEntry instance is passed to the FSASchedule instance via the addAlarm method


Client
: FSASchedule
: FSAXMLHelper
: AlarmManager
: Mapping
: Unmarshaller
1. FSASchedule( )
1.1. AlarmManager( )
2. configureXML(String, String)
2.1. FSAXMLHelper( )
2.2. parse(String, String)
2.2.1. Mapping( )
2.2.2. loadMapping(String)
2.2.3. Unmarshaller(Mapping)
2.3. addAlarm(final FSAScheduleEntry)

4.8.8 Reference

- JDring Schedule Framework

  http://webtools.dyade.fr/jdring

- Castor XML Parsing Framework

  http://www.castor.org/xml-mapping.html

- Sun Java Development Kit 1.2.2

  http://java.sun.com

4.8.9 Jar Files

The following Jars files are part of the ITA Scheduler Framework.

1. **JDRing Jar Files**
   - jdring.jar
2. **XMLHelper**
   - Xmlhelper.jar
3. **FSASchedule**
   - FSASchedule.3.0.jar
4. **Logging Jar Files**
   - Jakarta-oro-2.0.1
   - Jdom-B6
   - Protomatter-1-1-5.jar
   - Utility.jar
   - Xerces.jar
   - Xml.jar
5. **XML Document 's**
   - Rcs.xml (Logging)
   - Mapping.xml (Scheduler)
   - Schedule.xml (Scheduler)

4.8.10 FSASchedule Example

```
public static void main(java.lang.String[] args) {
        FSASchedule timer = new FSASchedule();
                //Add a new FSASchedule Object
        timer.addAlarm(55, 17, -1, -1, -1, -1, "gov.ed.fsa.ita.email.example.SMTPTester","main");
                //Add a FSAScheduleEntry to launch SMTPTester main at 17:55 every day.
        timer.configureXML(d:\www\map.xml,d:\www\schedule.xml);
                //Add another FSAScheduleEntry parsing a XML document.
}
```

4.9 Web Services

4.9.1 Overview

Web Services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. The invocation is done using a set of low overhead, open standard network and application protocols (e.g., UDDI, SOAP, XML), which will be discussed in more depth later in this document.

Web Services are loosely coupled services. The Web Services programming model is a service-oriented model based on the exchange of documents as opposed to arguments. The inputs, outputs, and return codes are all typically XML documents. By focusing solely on these document-based messages as interfaces, Web Services are completely language, platform, and object-model independent.

Web Services perform functions that range from simple requests to complicated business processes. At a minimum, a Web Service must provide information regarding available interfaces, the requirements on incoming messages, and the specifications of outbound messages. The implementation details of the services are completely masked from the external world.

The Web Services architecture is composed of many parts which are discussed in the following sections. The ITA team will implement the Single Object Access Protocol (SOAP) component of the Web Services architecture. The design for this implementation is discussed in section 4.9.3.3. The SOAP implementation will be used to transfer data and execute functionality between applications. Specifically, the implementation will be used by the Enterprise Architecture Integration (EAI) team to transfer data between financial institutions and FSA applications.

4.9.2 Benefits

Due to the loose coupling of services, there are a few key benefits to Web Services that are not applicable for client-server and other net-centric computing alternatives:

- The most prominent advantage of Web Services is they provide a straightforward, low entry cost mechanism for system-to-system interaction between organizations. The focus on standardization and agreement at the specification level supplies an opportunity to reduce costs and improve efficiencies across organizational boundaries

- Web Services are based on a set of industry standard protocols and technologies available on all platforms; therefore there is no dependency on any particular operating system, programming language, object-model protocol, or database

- The ability to "wrap" existing components/applications with Web Services technology and deploy them as Web Services, supports the reuse and extension of existing components and applications. This allows organizations to leverage the development investment in existing applications as well as reduce the time to deliver Web services for their organization, partners, and customers

- The elimination of a tightly coupled remote-procedure-call implementation provides the potential for Web Services to deliver standalone business functions that require a minimal amount of shared understanding in order to interoperate. The only information that must be shared is the SOAP XML documents which are easily and quickly created

- Web Services enables a distributed architecture that is data/service oriented as opposed to today's presentation-oriented model. Web Services provide an organization with an infrastructure that extends to deliver rich mobile applications by packaging information in a manner that allows local capabilities of the channel to be better leveraged

4.9.3 Technical Architecture

There are three main components to a web service: Service providers, service brokers, and service requestors. Service providers provide the services and maintain a registry that makes those services available. Service brokers assist the communication between the service providers and service requestors. Service requestors work with service brokers to discover web services and then invoke those services to create applications.

There are three main web services operations: Publish/unpublish, find, and bind. Publishing and unpublishing involves advertising services to a registry or removing those entries. The service provider contacts the service broker to publish or unpublish a service. The find operation is where service requestors describe the kinds of services they are looking for, and the service brokers deliver the results that best match the request. During the bind operation, the service requestor and the service provider negotiate so the requestor can access and invoke the services of the provider.

The following diagram depicts the Web Services Architecture:

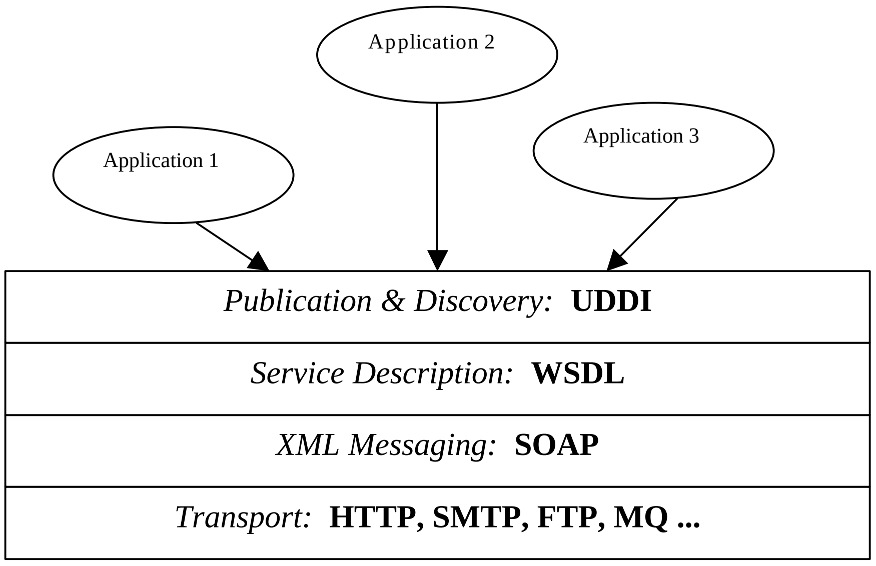

Application 2
Application 1
Application 3
Publication & Discovery: UDDI
Service Description: WSDL
XML Messaging: SOAP
Transport: HTTP, SMTP, FTP, MQ ...

Figure 1: Web Services Technical Architecture

4.9.3.1 WSDL

Web Services Description Language (WSDL) is an XML document used to define web services and the set of operations within each service that the server supports. The WSDL file also describes the format that the client must follow in requesting an operation. Since the WSDL file sets up requirements for both the server and the client, this file is akin to a contract between the two. The server agrees to provide certain services only if the client sends a property formatted SOAP XML request.

In a WSDL file, there are five primary elements used in defining the network service. The five elements are as follows:

- <types> element – defines the various data types used in exchanging messages
- <message> element – describes the messages being communicated
- <portType> element – identifies a set of operations and the messages involved with each of those operations
- <binding> element – specifies the protocol details for various service operations and describes how to map the abstract content of these messages into a concrete format
- <service> element – groups a set of related ports together

The following is an example of a WSDL file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="FooSample"
targetNamespace="http://tempuri.org/wsdl/"
xmlns:wsdlns="http://tempuri.org/wsdl/"
xmlns:typens="http://tempuri.org/xsd"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:stk="http://schemas.microsoft.com/soap-toolkit/wsdl-extension"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
<schema targetNamespace="http://tempuri.org/xsd"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
elementFormDefault="qualified" >
</schema>
</types>

<message name="Simple.foo">
<part name="arg" type="xsd:int"/>
</message>

<message name="Simple.fooResponse">
<part name="result" type="xsd:int"/>
</message>

<portType name="SimplePortType">
<operation name="foo" parameterOrder="arg" >
<input message="wsdlns:Simple.foo"/>
<output message="wsdlns:Simple.fooResponse"/>
</operation>
</portType>

<binding name="SimpleBinding" type="wsdlns:SimplePortType">
<stk:binding preferredEncoding="UTF-8" />
<soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="foo">
<soap:operation
soapAction="http://tempuri.org/action/Simple.foo"/>
<input>
<soap:body use="encoded" namespace="http://tempuri.org/message/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
<output>
<soap:body use="encoded" namespace="http://tempuri.org/message/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
```

```
</binding>

<service name="FOOSAMPLEService">
<port name="SimplePort" binding="wsdlns:SimpleBinding">
<soap:address location="http://carlos:8080/FooSample/FooSample.asp"/>
</port>
</service>
</definitions>
```

Figure 2. Sample WSDL file

4.9.3.2 UDDI

Universal Description, Discovery and Integration (UDDI) is a specification for information registries of web services, and hence part of the web services framework. UDDI is designed to provide a searchable directory of businesses and their web services. Each business registered with UDDI lists all its services and gives each of these services a type. This service type has a unique identifier and comes from a pool of well-known service types that are registered with UDDI. By having a pool of well-known service types, UDDI makes it possible to find out how to do electronic business with an organization. This is the primary advantage UDDI has compared to other Web-based business directories.

4.9.3.3 SOAP

4.9.3.3.1 Introduction

The Simple Object Access Protocol (SOAP) is a protocol based on the idea that a distributed architecture needs to exchange information. This protocol is lightweight, requiring a minimal amount of overhead and uses HTTP as the communication protocol. This allows the application to overcome challenging issues like firewalls (since firewalls usually allows HTTP transmissions) and avoid from having extraneous sockets listening on oddly numbered ports.

4.9.3.3.2 System Overview

There are three basic components to the SOAP specification: the SOAP envelope, a set of encoding rules, and a means of interaction between request and response (the invocation). The SOAP envelope supplies information about the message that is being encoded, including data relating to the recipient and sender, as well as details about the message itself. Before an application goes forward with processing a message, the application can determine information about a message, including whether it will be able to process the message. Distinct from the situation of standard XML-RPC (Remote Procedure Call) calls, SOAP's interpretation occurs in order to determine information about the message. A typical SOAP message can also include the encoding style, which assists the recipient in interpreting the message.

SOAP also brings a simple means of encoding user-defined datatypes. In remote procedure calls, encoding can only occur for a predefined set of datatypes: Those that are supported by an XML-RPC toolkit. Encoding other types requires modifying the RPC server and clients themselves. For SOAP, however, XML schemas can be used to easily specify new datatypes, and those new types can be represented in XML as part of a SOAP payload. Because of this integration with XML Schema, it is possible to encode any datatype in a SOAP message that can be logically described in an XML schema.

The SOAP invocation works by storing the invocation as a resident in memory. The invocation allows developers to set the target of the call, the method to invoke, the encoding style, and the parameters. It is more flexible than the XML-RPC methodology, whereby various parameters may be set explicitly, which are determined implicitly in XML-RPC.

4.9.4 Design Considerations

4.9.4.1 Assumptions and Dependencies

It is assumed that the SOAP framework functions in a J2EE application server environment. As the current production server for FSA is IBM's WAS v. 3.5, the framework is compiled using its required JDK version 1.2.2. It is assumed to be compatible with JavaServer Pages (1.1) and Java Servlet (2.2) specifications for this server. This framework is fully tested on the Sun Solaris 2.6 and HP-UX 11.0 operating systems. While this framework will be built using these product versions, it will be built in accordance with J2EE standards and to support product upgrades.

4.9.4.2 Goals and Guidelines

The goal of the SOAP framework is to provide a simple and robust framework that can be applied by FSA application teams, developing application in a Java environment. The SOAP framework implements the SOAP protocol put forth by the World Wide Web Consortium (W3C). The framework will abstract the syntax of the SOAP protocol from application developers.

The SOAP component of the web services framework is built using the Apache SOAP implementation. This implementation will be used for the following reasons:

- Apache SOAP is compatible with JDK 1.2.2
- Apache is a well-known name and its frameworks have been used before
- Apache SOAP is open source and is easily modified to suit an application's needs

- Apache Axis is the next release of Apache SOAP which contains additional features, and the code can be easily upgraded

4.9.4.3 Development Methods

This framework will be developed using general object-oriented software development techniques. The standard sequence diagram is provided in this document. This document assists developers who are unfamiliar with this framework.

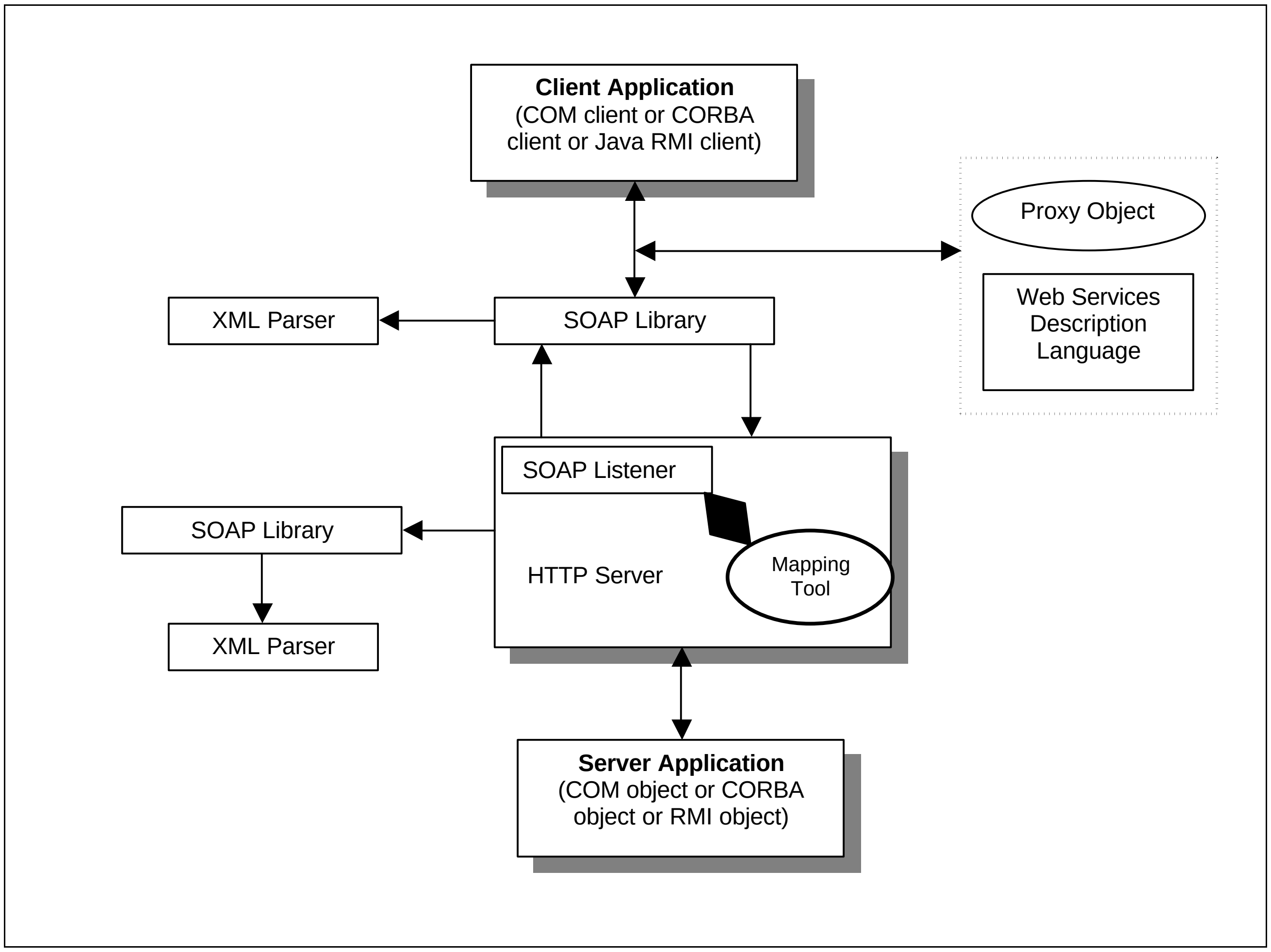
4.9.5 System Architecture

The SOAP framework provides the following services:

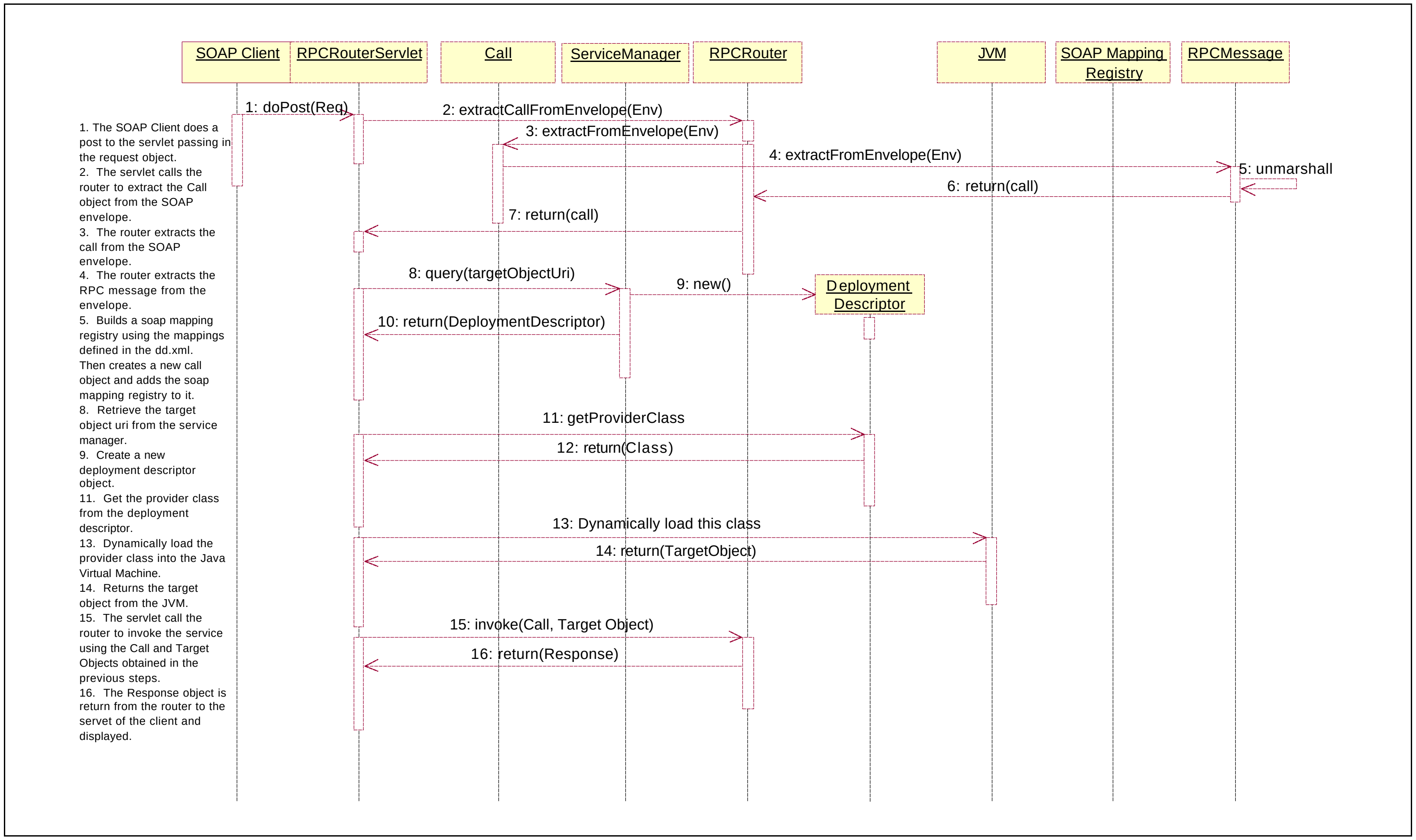
- SOAP Client – creates the SOAP message, sends it and receives the response from the service called

- SOAP Server – interprets the message from a SOAP Client, provides the service, and sends the message back to the SOAP Client

The following diagram depicts the technical architecture of the SOAP framework:


Client Application
(COM client or CORBA client or Java RMI client)
Proxy Object
Web Services Description Language
XML Parser
SOAP Library
SOAP Listener
SOAP Library
HTTP Server
Mapping Tool
XML Parser
Server Application
(COM object or CORBA object or RMI object)

4.9.6 Sequence Diagram



1. The SOAP Client does a post to the servlet passing in the request object.
2. The servlet calls the router to extract the Call object from the SOAP envelope.
3. The router extracts the call from the SOAP envelope.
4. The router extracts the RPC message from the envelope.
5. Builds a soap mapping registry using the mappings defined in the dd.xml. Then creates a new call object and adds the soap mapping registry to it.
8. Retrieve the target object uri from the service manager.
9. Create a new deployment descriptor object.
11. Get the provider class from the deployment descriptor.
13. Dynamically load the provider class into the Java Virtual Machine.
14. Returns the target object from the JVM.
15. The servlet call the router to invoke the service using the Call and Target Objects obtained in the previous steps.
16. The Response object is return from the router to the servet of the client and displayed.

4.9.7 References

- General Web Services Website

  https://www.webservices.org

- IBM Web Services Website

  http://www-106.ibm.com/developerworks/webservices/

- Apache SOAP Toolkit Website

  http://xml.apache.org/soap/index.html

- UDDI Website

  http://www.uddi.org/

Appendix A – ITA RCS R2.0 and FSA Applications Matrix

The following matrix details the current and potential usage of RCS services by FSA's applications.

	In Use	High Opportunity	Medium Opportunity	Low Opportunity
Web Conversation	Portals	Schools Portal, EIP		FAFSA, eCB, IFAP
Object Pooling		EAI	Portals	
Session		eAudit, FAFSA	Portals, Schools Portal, eCB, EIP	
FTP		Students & FP Portals		eCB, COD
Configuration		eAudit, Portals, FAFSA	Schools Portal, EIP, IFAP	eCB
XML Helper		COD		
Web Services		EAI (SOAP)		
JSP Tag Library		eAudit, Portals, Schools Portal	FAFSA, eCB, EIP, IFAP	
Scheduler		Students & FP Portals	Portals, Schools Portal, FAFSA	eCB, EIP, IFAP